TRABALHO DE GRADUAÇÃO

Reconhecimento de Placas Veiculares utilizando Deep Learning: Análise da influência de dados sintéticos no processo de reconhecimento

Saulo Cardoso Barreto

Brasília, Dezembro 2018

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

Reconhecimento de Placas Veiculares utilizando Deep Learning: Análise da influência de dados sintéticos no processo de reconhecimento

Saulo Cardoso Barreto

Relatório submetido como requisito parcial de obtenção de grau de Engenheiro de Controle e Automação

Banca Examinadora

Professor Flávio de Barros Vidal, CIC/UnB _____
Orientador

Professor Dibio Leandro Borges, CIC/UnB _____
Examinador Interno

MSc. Ana Paula G. S. de Almeida, PPMEC/UnB _____
Examinadora Externa

Brasília, Dezembro 2018

FICHA CATALOGRÁFICA

BARRETO, SAULO CARDOSO
Reconhecimento de placas veiculares utilizando Deep Learning: Análise da Influência do Uso de Dados Sintéticos no Processo de Reconhecimento / Saulo Cardoso Barreto; Orientador: Flávio de Barros Vidal

[Distrito Federal] 2018.

XV, 62p., 210 x297 mm (FT/UnB, Engenheiro, Controle e Automação, 2018). Trabalho de Graduação – Universidade de Brasília. Faculdade de Tecnologia.

1. Visão Computacional 2. Reconhecimento de placas
3. Redes Neurais

I. Mecatrônica/FT/UnB II. Reconhecimento de placas veiculares utilizando Deep Learning: Análise da influência do uso de dados sintéticos no processo de reconhecimento

REFERÊNCIA BIBLIOGRÁFICA

BARRETO, S. C. (2018) Reconhecimento de placas veiculares utilizando Deep Learning: Análise da influência do uso de dados sintéticos no processo de reconhecimento. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT. TG-$n°$14, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 77p.

CESSÃO DE DIREITOS

Saulo Cardoso Barreto

Campus Darcy Ribeiro, FT, Universidade de Brasília.

70919-970 Brasília – DF – Brasil.

Dedicatória

Dedico este trabalho aos meus pais, Tito Lívio e Alex Sandra

Saulo Cardoso Barreto

Agradecimentos

Inicialmente, gostaria de agradecer aos meus pais e irmãos, por todo o apoio dado, em especial à minha mãe por ter consciência da importância da educação e por isso sempre me incentivar na busca pelo conhecimento e ao meu pai por mostrar que independente do que se faça, que seja feito da maneira mais profissional possível.

Gostaria de agradecer também ao professor Flávio Vidal por ter aceito o convite para ser meu orientador, e por todos os conselhos e críticas ao projeto que apresento neste documento, bem como a instigação para desenvolver um trabalho consistente e bem fundamentado, contornando os diversos problemas enfrentados. Bem como ao Jorge Lambert e ao Departamento da Polícia Federal pela disponibilização da estrutura e de seu tempo nas etapas iniciais do projeto.

Agradeço também aos amigos que fiz na Graduação, cujo companheirismo foi essencial para finalizar todas as etapas e aos quais devo os melhores momentos desses últimos anos: Bruno, Lucas, Matheus, Yuji e Ricardo. Um agradecimento especial ao Túlio pelo apoio dado na última etapa desse trabalho.

Por fim, agradeço à CAPES por possibilitar a experiência do Ciência sem Fronteiras, na qual confirmei meu interesse pelo ambiente acadêmico e tive a oportunidade de me inserir em outra cultura e com isso ampliar meus horizontes.

Saulo Cardoso Barreto

RESUMO

O reconhecimento automático de placas de veículos tem sido alvo de diversos estudos, dada a sua aplicabilidade em situações reais: cobrança de pedágio, identificação de veículos em estacionamentos ou mesmo por questões de segurança em controle de veículos que cruzam fronteiras entre os países. O trabalho desenvolvido aqui consiste em retreinar tanto um modelo de reconhecimento de placas como um modelo de reconhecimento de objetos diversos, utilizando bases de dados sintéticas de placas no padrão brasileiro com variações de rotação, tamanho e ruído. Assim, a influência da utilização de placas sintéticas na acurácia de sistemas responsáveis por localizar placas reais, segmentar os caracteres e reconhecê-los foi avaliada e nos testes realizados houve um aumento da acurácia (em relação a um sistema treinado com placas reais) de três etapas: segmentação dos caracteres, reconhecimento de letras e reconhecimento dos números (2,54%, 1,09% e 2,49% respectivamente). Destaca-se a acurácia de 62,47% para a etapa de reconhecer os números, obtida por uma rede neural treinada exclusivamente com dados sintéticos e testada em placas reais.

Palavras Chave: redes neurais, reconhecimento de placas, base de dados sintética

ABSTRACT

The Automatic License Plate Recognition has been the subject of several studies, given its applicability in real situations: toll collection, identification of vehicles in parking lots or even for safety issues in vehicle control that cross borders between countries. The work developed here consists of retraining both a plate recognition model and a deep neural network for object detection, using synthetic plates databases in the Brazilian standard with variations of rotation, size and noise. Thus, the influence of the use of synthetic plates on the accuracy of systems responsible for locating real plates, segmenting the characters and recognizing them was evaluated and in the tests performed there was an increase in accuracy (considering a system trained with real plates) of three stages: character segmentation, letter recognition and number recognition (2.54 %, 1.09 % and 2, 49 % respectively). It stands out the accuracy of 62.47 % (in the number recognition step) obtained by a neural network trained exclusively with synthetic data and tested on real plates.

Keywords: Neural networks, license plate recognition, synthetic databases

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Siglas

ALPR	*Automatic License Plate Recognition*
GPU	*Graphic Processing Unit*
GAN	*Generative Adversarial Net*
CNN	*Convolutional Neural Networks*
MSE	*Mean Square Error*
ReLU	*Rectified Linear Unit*
OCR	*Optical Character Recognition*
mAP	*Mean Average Precision*
SVM	*Support Vector Machine*
R-CNN	*Region-Convolutional Neural Networks*
SPP-net	*Spatial Pyramid Pooling net*
FC	*Fully Connected*
SPP	*Spatial Pyramid Pooling*
RPN	*Region Proposal Network*
FPN	*Features Pyramid Network*
RoI	*Region of Interest*
YOLO	*You Only Look Once*
GT	*Ground Truth*
BB	*Bounding Box*

Símbolos Latinos

$\boldsymbol{x}$	*Entrada da rede neural*
x_n	*n-ésimo elemento da entrada da rede neural*
$\boldsymbol{y}$	*Mapeamento realizado por uma rede neural*
$tanh$	*Tangente hiperbólica*
max	*Função máximo*
h_j	*Camadas ocultas da rede neural*
o_j	*Entrada da função de ativação*
w_{ji}	*Peso associado ao i-ésimo neurônio*
W	*Matriz de parametrização dos pesos*
$E(W)$	*Função de Perda associada ao erro dos pesos*
$\widetilde{E}(W)$	*Função de Perda com o decaimento de pesos adicionado*
$f^*(x)$	*Valores esperados para a saída da rede neural*
$f(\boldsymbol{x}, \theta)$	*Valores estimados para a saída da rede neural*
^{i}	*Índice de um elemento num conjunto de dados*
s	*stride*
S	*Tamanho do stride*
B	*Quantidade de Bounding Boxes*
$Pr(Object)$	*Probabilidade de Existência de um Objeto*
IOU_{pred}^{truth}	*Intersection over Union de duas regiões*
C	*Probabilidades Condicionais de Classes*
i	*Índice da célula analisada na arquitetura YOLO*
j	*Índice da bounding box analisada na arquitetura YOLO*
x_i	*Coordenada do centróide do GT*
$\hat{x}_i$	*Coordenada estimada do centróide*
y_i	*Coordenada do centróide do GT*
$\hat{y}_i$	*Coordenada estimada do centróide*
w_i	*Largura da bounding box do ground truth*
$\hat{w}_i$	*Largura estimada da bounding box prevista*
h_i	*Altura da bounding box do ground truth*
$\hat{h}_i$	*Altura estimada da bounding box prevista*
C_i	*Índice de confiança esperado sobre a existência de um objeto*
$\hat{C}_i$	*Índice de confiança obtido sobre a existência de um objeto*
p_i	*Probabilidade condicional de class esperada*
$\hat{p}_i$	*Probabilidade condicional de class obtida*

Símbolos Gregos

θ	*Parâmetros gerados pelo processo e aprendizado*
∇	*Gradiente de uma função*
Δ	*Incremento de uma variável*
μ	*Taxa de aprendizado de uma rede neural*
α	*Momentum de uma rede neural*
λ	*Parâmetro de decaimento de pesos*
λ_{coord}	*Fator de multiplicação para influência do erro de localização*
λ_{obj}	*Fator de multiplicação da função de perda (YOLO) quando existe um objeto*
λ_{noobj}	*Fator de multiplicação da função de perda (YOLO) quando não existe um objeto*

Capítulo 1

Introdução

"A ciência nunca resolve um problema sem criar pelo menos outros dez." (George Bernard Shaw)

1.1 Contextualização

A tarefa de localizar e identificar placas de carro em tempo real ou em imagens já capturadas, tem recebido cada vez mais atenção em pesquisas de Visão Computacional devido à sua importância e grande variedade de aplicações, tais como: investigação de acidentes de trânsito [25], violações de pedágios [26], análise de imagens em cenas de crime [27], verificação automatizada de tickets de estacionamento [28] e na fiscalização dos limites de velocidades [29]. Diversos algoritmos e abordagens vem sendo utilizados para tentar resolver esse problema, como pode ser visto na revisão de trabalhos na área publicada em 2013 em [30], ou o trabalho desenvolvido por Anagnostopoulos *et al.* em [31], que analisou mais de 100 artigos e métodos publicados, agrupando-os por similaridade na solução proposta.

Define-se ALPR (*Automatic License Plate Recognition*) como um sistema que tem como objetivo realizar as tarefas descritas anteriormente para que se obtenha um resultado satisfatório na aplicação final: localizar uma placa e reconhecer cada caractere.

Como indicado por Li *et al.* [32], uma imagem de entrada de um sistema ALPR é capturada sob a influência de diversos fatores: resolução da câmera, orientação tanto do veículo quanto da câmera, presença de luz no ambiente no momento da captura, velocidade do obturador da câmera, condições climáticas entre outros. Dada a falta de uniformidade das bases de dados existentes, as limitações de cada sistema desenvolvido e as variações de placas de cada região é inapropriado comparar acurácias de diferentes sistemas sem considerar a complexidade dos fatores que influenciam os resultados de cada um [31].

O processo para reconhecer uma placa pode ser divido em três etapas, ilustradas na Figura 1.1.:

1. Detecção da Placa - Localizar a região de uma imagem correspondente à placa do veículo
2. Segmentação dos Caracteres - A partir da imagem da placa localizar e segmentar os caracteres

3. Reconhecimento dos Caracteres - A última etapa consiste em reconhecer cada letra e número da placa.

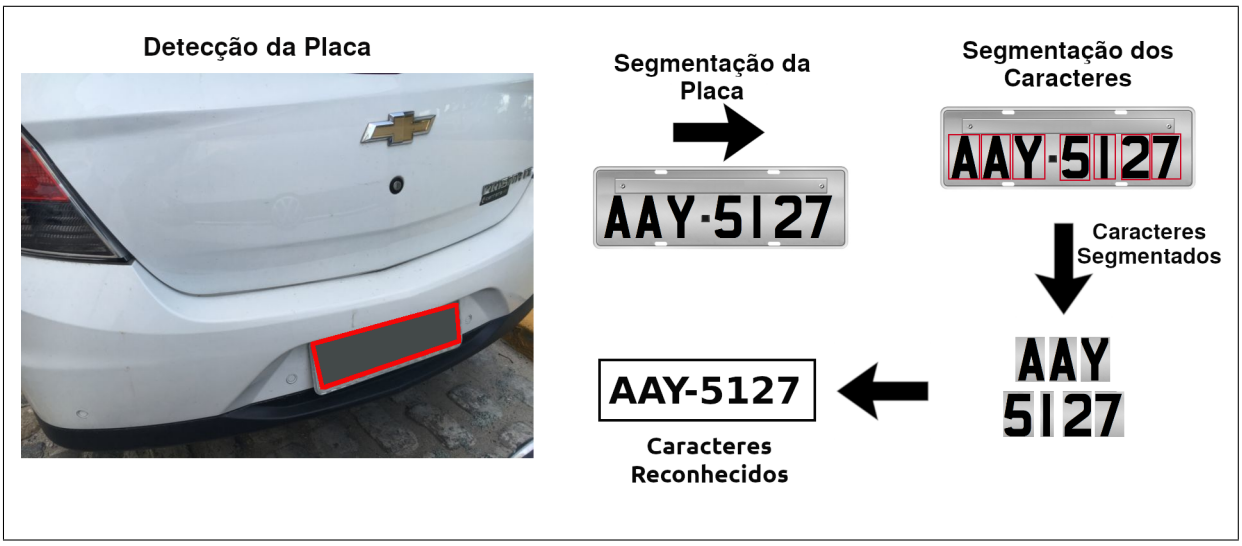


Figura 1.1: Etapas de um sistema ALPR.

Para a primeira etapa, vários métodos de extração já foram utilizados: baseado na textura [33], localização e existência de caracteres [34], utilizando informações de bordas [35] ou possuindo um conhecimento prévio dos caracteres [36].

No segundo passo, a placa extraída anteriormente pode apresentar problemas de rotação ou brilho. Alguns métodos também já foram desenvolvidos para contornar o problema e propor correções: i) transformação bilinear [37], ii) método de mínimos quadrados foram utilizados para remover a inclinação tanto horizontal como vertical nas placas [38].

A saída do sistema é a sequência de caracteres reconhecida, porém, alguns problemas são esperados nesta última etapa: uma mesma placa pode apresentar caracteres de tamanhos diferentes, cada país ou região possui fontes e cores diferentes, ou mesmo letras e números completamente ilegíveis (imagem com ruído, caracteres cobertos por lama, poeira, etc). Para essa última etapa, é possível comparar cada caracter com uma base de dados e definir a similaridade com cada símbolo da mesma como em [39] ou utilizando *features* extraídas [40].

Técnicas de *machine learning* tem sido utilizadas em diversas áreas de estudo: desde a classificação de células baseando-se em sua morfologia [41] ou até mesmo para identificação de lesões em radiografias [42]. O mesmo ocorre para as 3 etapas do reconhecimento de uma placa. O maior desafio para a ALPR, dado o avanço da capacidade de processamento computacional com o advento das GPUs [2], e que a impede de alcançar resultados próximos aos de reconhecimento de objetos, por exemplo, é a indisponilidade de uma base de dados anotada de larga escala, dada a dificuldade em se coletar e categorizar um conjunto de dados com placas reais.

Numa situação em que não é possível se obter uma base de dados de larga escala, emprega-se o método de Ampliação de Dados para criar uma base de dados sintética: uma base de dados pode ser expandida a partir de transformações que preservem as classes de treinamento; são realizados

cortes, rotações, mudanças de perspectiva, espelhamento [43], adição de *background* [44] ou até mesmo a utilização de uma outra rede classificada como GAN (*Generative Adversarial Net*) [10]

Um maior detalhamento de todos os métodos citados nessa Seção é realizado no próximo capítulo.

1.2 Escopo do Projeto

Sabendo que as placas dos veículos apresentam inúmeros padrões diferentes, a depender do país em análise, o projeto desenvolvido neste trabalho se limita às placas no padrão brasileiro vigente em 2018.

- **Padrão de caracteres:** No Brasil, todas as placas apresentam o mesmo padrão, a diferença entre os estados fica restrita apenas às letras e aos números utilizados que são exclusivos de cada região.

  XXX-YYYY

  onde X é uma letra, Y é um número e um hífen separa ambos os blocos de caracteres. No Distrito Federal, por exemplo a última sequência liberada em 2014 compreende as placas de OZW-0001 a PBZ-9999

- **Cores das placas:** Dependendo da utilização do carro, a placa do mesmo pode ser de diferentes cores: fundo vermelho com caracteres brancos (veículos utilizados para transporte remunerado), fundo branco com caracteres pretos (veículos do poder público), fundo azul com caracteres brancos (veículos em missão diplomática), fundo preto com caracteres brancos (veículos de colecionadores), fundo verde com caracteres brancos (veículos de teste das montadoras), fundo cinza com caracteres pretos (carros particulares). Dada a grande quantidade de variações existentes e que a maioria das placas brasileiras apresentam o último padrão citado, este será o modelo adotado no projeto.

- **Fonte:** Até 2008, a fonte utilizada nas placas brasileiras era a *DIN Mittelschrift*, após esse ano uma resolução do Contran [45] alterou o sistema de placas de identificação e a fonte que passou a ser utilizada foi a *Mandatory*, e essa foi a fonte utilizada na criação da base de dados e percebida nas bases de dados reais.

É importante considerar ainda as distorções associadas ao dispositivo de captura ou mesmo ao ambiente capturado, todos os aspectos citados a seguir foram elaborados a partir da observação da base de dados real utilizada.

- **Tamanho:** A imagem capturada pode apresentar diversos tamanhos, bem como a placa, que pode corresponder a diferentes proporções de toda a imagem de estudo (A Figura 4.6(a), presente na base utilizada, tem dimensões 212x64 pixels).

- **Iluminação:** Um ambiente com pouca luz pode comprometer a qualidade da imagem capturada, como ilustrado na Figura 4.6(c).
- **Localização:** A placa pode estar localizada em qualquer posição na imagem capturada.
- **Ângulo e rotação:** As diversas imagens da base de dados apresentam placas capturadas de diferentes ângulos, como pode ser visto na Figura 4.6(d), ou mesmo rotacionadas.
- **Ruído:** Algumas placas possuem poeira ou foram capturadas por câmeras em movimento (ou até mesmo de baixa qualidade), justificando a trepidação ou enevoamento.
- **Outros aspectos:** Algumas placas apresentam desgaste natural (Figura 1.2(f).), associado ao tempo de circulação, outras estão em escala de cinza (Figura 1.2(e).), ou estão até mesmo amassadas.

(a) (b) (c) (d) (e) (f)

Figura 1.2: Exemplos de distorções nas imagens capturadas: (a) exemplo de imagem com tamanho reduzido; (b) exemplo de placa ocupando quase a totalidade da imagem; (c) captura efetuada à noite; (d) captura efetuada em angulação diferente; (e) exemplo de imagem em escala de cinza; e, (f) exemplo de placa que apresenta desgaste [1].

1.3 Objetivos

1.3.1 Objetivo Geral

Analisar a influência do treinamento de uma rede neural convolucional, realizado com uma base de dados sintética, no reconhecimento de placas reais.

O objetivo geral do trabalho é verificar a capacidade de reconhecimento de placas reais de uma rede neural treinada com uma base de dados sintética, bem como verificar a acurácia de uma rede previamente treinada com imagens reais, retreinada com placas artificiais (*transfer learning* e *fine-tuning*).

1.3.2 Objetivo Específico

Comparar o desempenho de diferentes arquiteturas de redes neurais de reconhecimento de objetos, retreinadas para realizar o reconhecimento de placas de carros.

Tem-se como objetivo, comparar arquiteturas desenvolvidas e treinadas para o reconhecimento de objetos [17, 46], testando-as com uma base de dados de placas reais que não foram utilizadas em qualquer etapa dos treinamentos. Assim, pretende-se realizar um retreinamento com bases de dados artificiais nessas redes que foram treinadas para reconhecer pessoas, animais, objetos, etc.

1.3.3 Hipótese 1

O treinamento de uma rede neural com placas de veículos exclusivamente sintéticas possibilita a correta localização da placa numa imagem e a segmentação e reconhecimento dos caracteres em placas reais.

Considerando-se as dificuldades de se obter bases de dados reais com placas veiculares anotadas, a possibilidade de se treinar uma rede e obter uma alta acurácia no reconhecimento dos caracteres, utilizando apenas placas artificiais, mostraria-se como um promissor resultado na área de Visão Computacional.

1.3.4 Hipótese 2

Uma rede neural previamente treinada apenas com placas reais ou com diversas classes de objetos, tem seu desempenho e acurácia afetados a partir do transfer learning obtido de um novo treinamento com placas sintéticas.

Tendo como base uma rede neural treinada num *framework* de reconhecimento de objetos, verificar a influência na acurácia do sistema ao realizar um novo treinamento, partindo dos pesos iniciais do treinamento prévio, porém utilizando dados artificiais como entrada.

1.4 Resultados e Contribuições

O trabalho desenvolvido nesse manuscrito comprovou que o treinamento realizado com bases de dados sintéticas pode aumentar a acurácia de redes neurais previamente treinadas para um sistema *ALPR* apenas com placas veiculares reais (Seções 5.2 e 5.3).

Os resultados obtidos com redes treinadas para a tarefa de localização e reconhecimento de objetos, retreinadas para a localização de placas de carros também se destacam, dada a acurácia obtida para um número relativamente menor de iterações em relação ao treinamento da rede profunda (Subseção 5.1.3).

Assim, acredita-se que a contribuição da Metodologia proposta e dos Resultados obtidos consiste em avaliar a utilização de base de dados artificiais para os casos em que não é possível obter uma base de dados real de larga escala, obtendo redes neurais com um desempenho satisfatório com uma definição correta de parâmetros.

1.5 Apresentação do Documento

Esse documento será dividido em seis capítulos. O Capítulo 2 aborda a fundamentação teórica dos conceitos utilizados no desenvolvimento do projeto relacionados ao aprendizado de máquina. O terceiro capítulo apresenta pesquisas relevantes na área de *ALPR*, bem como os vários métodos já implementados e seus respectivos resultados. O Capítulo 4 apresenta a Metodologia proposta: arquiteturas utilizadas, *framework*, bases de dados utilizadas, bem como todas as decisões tomadas ao longo do Desenvolvimento. O quinto capítulo apresenta os resultados obtidos e os analisa na Seção de Discussão. O último capítulo conclui o trabalho analisando perspectivas futuras a partir dos resultados obtidos.

Capítulo 2

Referencial Teórico

2.1 *Deep Learning* e Aprendizagem de Máquina

Fazer com que o computador execute uma sequência de instruções bem definidas e estruturadas não é uma tarefa difícil. Até mesmo programá-lo para executar ações modeladas por expressões matemáticas que envolvem probabilidade e estatística já se provou um desafio alcançado: em 1997 o campeão mundial de xadrez, Gasparov, perdeu para um computador disputando a modalidade em que era visto como melhor do mundo [47]. O desafio se tornou então, ensinar o computador a reconhecer padrões, tomar decisões e realizar previsões em tarefas de difícil definição formal para um ser humano. A solução adotada na área da Ciência da Computação foi possibilitar ao computador aprender pela experiência, utilizando o conceito de camadas, para que a abstração fosse realizada passo-a-passo, para que, em larga escala, ele seja capaz de aprender conceitos complexos [2].

Partindo desse conceito, de camadas simples que sobrepostas possibilitam o entendimento de um mundo complexo, começou-se a utilizar o termo *deep learning*: a profundidade está associada a um grafo formado pela composição de uma camada sobre a outra. Enquanto adquirir a capacidade de extrair informações relevantes que caracterizam um objeto de estudo e a partir daí desenvolver conhecimento está associada ao conceito de *Machine Learning* (ou aprendizado de máquina) que depende, essencialmente de como os dados são representados e apresentados ao computador [48].

2.2 *Deep Feedforward Networks*

Para compreender o funcionamento de uma rede neural convolucional ou *convolutional neural networks (CNN)*, é primeiramente necessário entender as definições formais básicas associadas ao conceito. Uma rede neural direta - ou em avanço (*Deep Feedfoward Networks*) - pode ser vista como uma rede de múltiplas camadas na qual a informação se propaga em apenas um sentido (as redes neurais que incluem conexões de *feedback* entre suas camadas são chamadas de redes neurais recorrentes) [22], assim, pode ser representada como um grafo acíclico e direcionado, como mostrado na Figura 2.1.

Define-se uma rede neural como um modelo de *deep learning* que realiza o mapeamento de uma entrada $\mathbf{x}$ para uma categoria $\boldsymbol{y}$, dado um classificador

$$\boldsymbol{y} = f^*(\boldsymbol{x}). \tag{2.1}$$

O objetivo da rede é aproximar a função f^* pelo mapeamento

$$\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{\theta}). \tag{2.2}$$

Na qual $\boldsymbol{\theta}$ representa os parâmetros gerados pelo processo de aprendizado da rede, visando a maior aproximação possível com o classificador original y . Analisando o conceito, a justificativa do uso do termo rede dá-se pela utilização de várias funções $(f^{(1)}, f^{(2)}, f^{(n)})$, conectadas em cascata para formar a função $f(\boldsymbol{x})$. Cada uma dessas funções é uma camada oculta da rede, que pode ser ainda de outros tipos [2]:

- Camada de Entrada: Possui as x_n entradas da rede que são ligadas por pesos de conexões às camadas ocultas.
- Camadas Ocultas (*Hidden Layers*): São as camadas responsáveis por extrair as *features* da imagem, determinando quais informações são relevantes no processo de treinamento, para cada categoria, baseando-se no algoritmo implementado. Na Figura 2.1, os nós h_1 e h_2 representam essas camadas.
- Camada de Saída: A última camada da rede neural apresenta o resultado obtido para a aproximação de $f^*(\boldsymbol{x})$ alcançada por $f(\boldsymbol{x})$.

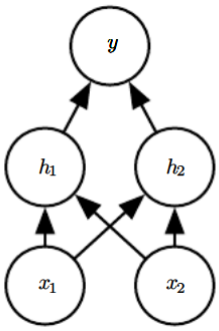


Figura 2.1: Exemplo de rede neural direta [2].

Cada nó da Figura 2.1 representa um neurônio: a unidade básica de uma *CNN*, responsável por calcular a combinação linear de suas respectivas entradas e representada na Figura 2.3. Um neurônio h_j calcula sua saída como descrito pela Equação 2.4, na qual o_j é calculado como:

$$o_j = \sum_{i=1}^{n} w_{ji} x_i + b_j. \tag{2.3}$$

Sendo w_{ji} o peso da conexão entre x_i e h_j, b_j é um deslocamento linear (em inglês *bias*) e o_j é o valor que será a entrada de uma função de ativação (f), responsável por calcular o valor final de h_j.

$$h_j = f(o_j) = f(\sum_{i=1}^{n} w_{ji}x_i + b_j). \tag{2.4}$$

As funções de ativação f comumente utilizadas são descritas na subseção 2.2.3. O resultado final vai depender dos pesos e deslocamentos lineares de uma rede e pode ser representado como $y = f_W(x)$, na qual x é a entrada e W é uma matriz que foi parametrizada para se ajustar aos dados.

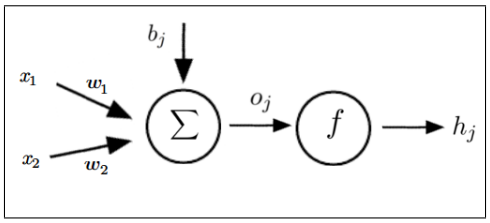


Figura 2.2: Representação da unidade básica da rede neural.

2.2.1 Erro e Função de Perda (*Loss Function*)

Dado um conjunto de treinamento, é necessário calcular o erro ao longo do processo de aprendizado, a fim de que no final tenha-se a aproximação de x pelo classificador y. A função de perda calcula esse erro para que os parâmetros da matriz W sejam ajustados, aumentando o índice de acertos da classificação.

Para os casos em que o problema pode ser resolvido por meio de uma regressão (a rede neural é responsável por fornecer um resultado numérico), a função mais comumente utilizada é a do Erro Quadrático Médio (*Mean Square Error - MSE*), sabendo que $f(\boldsymbol{x};\theta)$ e $f^*(\boldsymbol{x})$ são respectivamente os valores estimados e esperados para uma dada entrada x, na qual N representa a quantidade de elementos do conjunto:

$$E(W) = \frac{1}{N}\sum_{i=1}^{N}(f^*(\boldsymbol{x}) - f(\boldsymbol{x};\theta)). \tag{2.5}$$

Caso a saída da rede neural seja uma classificação dentre muitas possibilidades, a fórmula utilizada para a função de perda é a de Entropia Cruzada (*Cross Entropy*):

$$E(W) = \frac{1}{N}\sum_{i=1}^{N}[f^*(\boldsymbol{x})log(f(\boldsymbol{x};\theta)) + (1 - f^*(\boldsymbol{x}))log(1 - f(\boldsymbol{x};\theta))]. \tag{2.6}$$

No *framework* utilizado nesse trabalho [46], uma função de perda específica é utilizada e detalhada na Seção 4.

2.2.2 Retropropagação (*Backpropagation*)

O objetivo de uma rede neural é, a partir dos parâmetros definidos na matriz W, minimizar o erro do sistema. Esse processo é feito geralmente por meio de retropopagação: considerando uma rede neural que apresente pelo menos uma camada oculta, o erro para um certo momento é propagado para uma camada anterior, na qual os pesos e deslocamento linear são alterados, de forma a reduzir o erro. A função que modifica os pesos é definida como **Função de Otimização** [49], que por sua vez, calcula o gradiente (derivadas parciais) da função de perda em relação aos pesos, modificando-os no sentido oposto ao do gradiente calculado.

O cálculo do gradiente da função $E(W)$ é feito computacionalmente de maneira iterativa. Uma mudança relativamente pequena é feita na matriz W (ΔW), assim:

$$W^{i+1} = W^i + \Delta W^i. \tag{2.7}$$

Na qual i representa a i-ésima iteração e o valor de ΔW^i é definido pela função de otimização. Essa mudança adicionada altera o valor da função de perda, como mostrado na Equação 2.8, na qual se verifica que a função de perda se aproxima de zero quando $\nabla E(W)$ se aproxima de zero, sabendo que quando o gradiente $\nabla E(W)$ se torna 0, $\Delta E(W)$ também passa a valer 0, indicando que um mínimo local foi alcançado [22].

$$\Delta E \approx \Delta W^i \nabla E(W). \tag{2.8}$$

No fim dos anos 90, tanto a retropropagação quanto as redes neurais não receberam atenção da comunidade envolvida com aprendizado de máquina: pensava-se que a extração de *features* sem conhecimento prévio fosse inviável. Mas hoje, diversos resultados de trabalhos teóricos e empíricos provam que os resultados obtidos com os mínimos locais não caracterizam um problema na convergência de redes neurais para a maioria das aplicações, como constatado por LeCun *et al* [48].

2.2.3 Funções de Ativação

A função de ativação de um neurônio artificial tem como objetivo definir se a informação recebida é relevante ou não para a classificação do dado (caso seja relevante, a informação é utilizada como entrada do próximo neurônio, caso não seja, o neurônio não é ativado). Essa função pode ser linear, que apesar de apresentar solução simples, não é capaz de resolver problemas mais complexos como a classificação de objetos numa cena. Assim, funções não-lineares são utilizadas para essa etapa, para que seja possível se aproximar de padrões complexos de dados [22].

- Sigmóide (*sigmoid*):

$$f(x) = \frac{1}{(1 + e^{-x})}. \tag{2.9}$$

Sendo uma função continuamente diferenciável (Equação 2.9), a sigmóide é utilizada em redes neurais em que se deseja ter apenas propagação positiva (os valores vão de 0 a 1), e o seu formato em S faz com que os valores de $f(x)$ sejam mapeados, em sua maioria para 1 ou 0. Apresenta vantagem em relação a função identidade por apresentar comportamento não-linear. Porém, quando o gradiente se torna muito pequeno, o mesmo tende a zero, o que representa que a rede não está convergindo [22].

- Tangente Hiperbólica ($TanH$)

$$tanh(x) = \frac{2}{(1 + e^{-2x}) - 1}. \tag{2.10}$$

Similar à função sigmóide, resolve o problema da limitação de propagação exclusivamente positiva (a saída assume valores de -1 a 1), porém apresenta a mesma limitação: o gradiente nos extremos tende a desaparecer [22].

- Unidade Linear Retificada ($ReLU$)

$$f(x) = max(0, x). \tag{2.11}$$

Por apresentar convergência rápida quando comparada às outras funções de ativação, essa função tem sido amplamente utilizada em redes neurais. Não apresenta o problema do gradiente tendendo a desaparecer e reduz o custo computacional por não realizar o cálculo de exponenciais. Porém, apresenta a limitação citada anteriormente: sua saída apresenta apenas valores positivos [22].

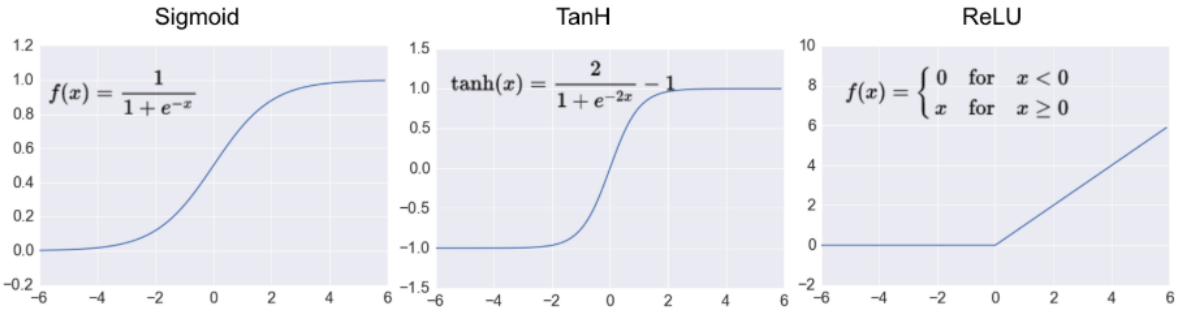


Figura 2.3: Representações das funções de ativação [3].

2.2.4 Parâmetros de Aprendizado de Máquina

Para que exista um método para determinar a taxa de convergência de uma rede neural, é necessário definir alguns parâmetros que influenciam diretamente tanto na acurácia quanto no processo de aprendizado.

- Taxa de aprendizado (*Learning Rate* - μ): o valor dessa variável define o quanto os pesos serão ajustados em cada iteração. Quanto menor for esse valor, mais lento será o processo

de retropropagação e poderá até mesmo demorar muito tempo em uma região de *plateau*. Porém, caso uma taxa de aprendizado muito alta seja utilizada, o mínimo local desejado pode ser ultrapassado pela iteração, e por conseguinte, nunca alcançado. Assim, uma taxa de aprendizado baixa geralmente é utilizada mesmo que demande maior tempo computacional, garantindo uma estabilização no processo de convergência [22].

$$\Delta W^{i+1} = \mu \nabla E(W^i). \tag{2.12}$$

- *Momentum* - α: Esse parâmetro define o efeito de mudanças passadas dos pesos na direção atual do movimento no espaço dos pesos, ao passo que assume valores entre 0 e 1, representando que os pesos dependem apenas do gradiente da função de perda ($\alpha = 0$) ou só depende do último ajuste de peso realizado ($\alpha = 1$). O valor adotado para essa variável costuma ser 0.9 [50].

$$\Delta W^{i+1} = (1 - \alpha)\mu \nabla E(W^i) + \alpha \Delta W^{i-1}. \tag{2.13}$$

- Decaimento dos Pesos (*Weight Decay* - λ): Essa variável é responsável por regularizar a função de perda. Uma função Gaussiana com média 0 é adicionada à função de perda, juntamente com a matriz de pesos (W). A nova função de perda ($\widetilde{E}(W)$) passa a ser [22]:

$$\widetilde{E}(W) = E(W) + \frac{\lambda}{2} W^2. \tag{2.14}$$

Aplicando essa equação na equação 2.13, obtém-se:

$$\Delta W^{i+1} = (1 - \alpha)\mu \nabla E(W^i) + \alpha \Delta W^{i-1} - \mu \lambda W^i. \tag{2.15}$$

Assim, evidencia-se, ao se analisar o último termo da equação 2.15, que quanto maior for o peso, maior será a penalidade aplicada (o peso decai proporcionalmente ao seu tamanho), limitando o número de parâmetros livres e evitando o sobreajuste [22].

2.2.5 Convolução

A operação matemática denominada convolução pode ser formalmente definida [22] conforme a Equação 2.16.

$$(f * g)(t) = \int_0^t f(\tau) * g(t - \tau) d\tau \tag{2.16}$$

Na qual calcula-se a convolução das funções f e g, resultando numa terceira função. Sabendo que essa é a definição de convolução no tempo contínuo, define-se ainda a convolução num conjunto de dados discreto (tal como uma imagem, definida pelos seus pixels, que não caracterizam continuidade).

$$(f * g)[n] = \sum_{i=-I}^{I} f[n-i] * g[i] \tag{2.17}$$

2.3 Redes Neurais Convolucionais

Ainda em 1998, LeCun propôs a utilização de redes neurais convolucionais para reconhecimento de imagens, voz e séries temporais [51]. O que a diferencia dos outros tipos de redes neurais é a utilização da operação de convolução em substituição à multiplicação de matrizes, em pelo menos uma de suas camadas. Baseia-se ainda no modelo biológico do córtex visual dos gatos que possui células com sub-regiões pequenas que funcionam como um tipo de filtro e tem diferentes respostas para cada tipo de célula, como observado por Hubel e Wiesel [52].

A estrutura de uma *CNN* se baseia em quatro conceitos essenciais, que se aproveitam de propriedades dos sinais naturais: conexões locais, pesos compartilhados, *pooling* e o uso de múltiplas camadas [48].

2.3.1 Campo de Conexões Locais

Cada pixel de uma imagem é utilizado como entrada de um neurônio numa rede neural convolucional [53]. A Figura 2.1 é um exemplo de rede neural completamente conectada: todos os neurônios de uma camada estão conectados a todos os neurônios da camada seguinte. Ao utilizar uma imagem pequena, o número de conexões não seria um problema, porém, cada pixel adicionado na imagem faz com que o número de conexões aumente exponencialmente.

Para contornar esse problema, as redes neurais convolucionais utilizam o conceito de **campo de conexões locais** (*local receptive fields*), considerando que não é necessário conectar todos os neurônios de uma camada à camada seguinte, dado que um pixel de uma imagem não está relacionado com todos os outros pixels da mesma, apenas com alguns pixels que estão em sua proximidade.

Pode-se usar um filtro de n x n pixels em torno do pixel analisado, definido como o *kernel* da camada, que ainda se desloca s (*stride*) pixels em cada iteração, como ilustrado na Figura 2.4 na qual um *kernel* 3 x 3 foi utilizado para uma imagem de entrada com dimensões 5 x 5 que ainda possui um *padding* (pixels transparentes em torno da imagem azul) com tamanho $p = 1$ adicionado para que a informação de borda não se perca na convolução. O *stride* nesse caso é igual a 1 [4].

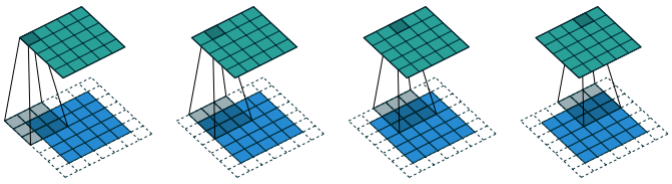


Figura 2.4: Representação das conexões locais [4].

A Figura 2.5 representa como um mapa de *features* é criado com um *kernel* 3 x 3, utilizado para uma imagem de entrada com dimensões 5 x 5, *padding* igual a 1, *stride* igual a 2, gerando um mapa de 9 *features* (3 x 3) representado em verde na subfigura superior esquerda. Ao se aplicar a operação de convolução, recursivamente, ao mapa obtido, têm-se um mapa mais profundo, representado em laranja, na qual cada feature tem um campo de conexões locais de tamanho 7 x 7. As subfiguras localizada à direita representam explicitamente quais pixels formam cada região definida por uma feature, além do tamanho dessa região

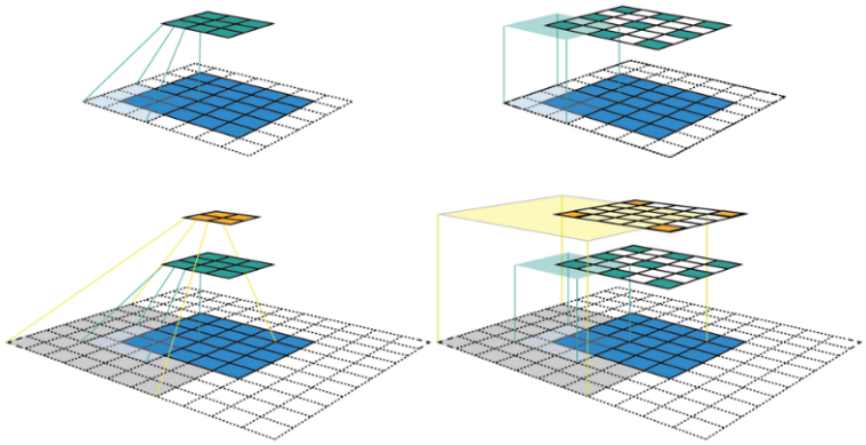


Figura 2.5: Representação do mapa de *features* [5].

2.3.2 Pesos Compartilhados

No início da Seção 2.2, foi demonstrado que numa rede neural cada neurônio possui um peso w_{ji} associado. Numa rede neural convolucional, todos os pesos e deslocamentos lineares são compartilhados e utilizados em toda a imagem [54]. Assim, o mesmo filtro é aplicado para cada n x n pixels, e uma *feature* pode ser localizada em qualquer região da imagem de entrada, conferindo assim às *CNN's* as propriedades de *invariância à translação* e *invariância à rotação*, dependendo do filtro utilizado. Considerando que a matriz $\boldsymbol{W}$ passa a ter parâmetros compartilhados por todos os neurônios, e não um w_{ij} para cada um, o número de parâmetros utilizado por uma rede neural decresce substancialmente ao se utilizar o modelo convolucional.

2.3.3 Camadas de *Pooling* e *ReLU*

É uma prática comum, ao se estabelecer a arquitetura da *CNN*, adicionar uma camada de *pooling* entre camadas de convolução, já que esse tipo de camada reduz consideravelmente o tamanho dos dados de entrada das camadas subsequentes bem como os parâmetros calculados computacionalmente [55]. A Figura 2.6 ilustra a aplicação de um filtro **max** 2 x 2 aplicado a uma imagem com dimensões 4 x 4, utilizando-se *stride* igual a 2: o filtro faz com que uma imagem de 16 pixels pixels seja representada como apenas 4 pixels, que possuem os maiores valores dentre os que foram considerados pelo *kernel*.

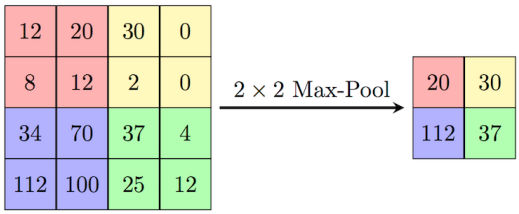


Figura 2.6: Representação de aplicação de uma camada *max-pooling* [6].

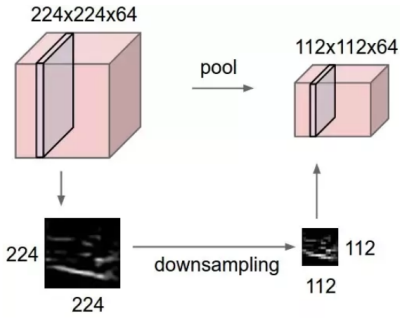


Figura 2.7: Exemplo de *pooling* numa imagem real [7].

A função *ReLU*, anteriormente citada na Subseção 2.2.3 , não altera o tamanho da imagem de entrada, mas aumenta a não-linearidade da rede ao realizar o truncamento de cada elemento da entrada (Equação 2.11). A intenção desse aumento é se aproximar do mundo real: uma imagem real também apresenta diversas não-linearidades (transição dos pixels, cores, bordas) [56].

A Figura 2.10. representa uma imagem submetida à uma camada de convolução que resulta num mapa de *features*, que em seguida é submetido à uma camada *ReLU*, responsável por criar um mapa retificado, no qual não se notam mudanças graduais mas abruptas.

Figura 2.8: Imagem submetida à convolução e em seguida à uma camada ReLU [8].

2.4 Capacidade de Generalização

O treinamento de redes neurais profundas tem obtido grande sucesso, devido em parte à crescente disponibilidade de bases de dados anotadas de larga escala. O desafio de reconhecer objetos é um exemplo de tarefa que alcançou excelenetes resultados: *mAP*(*Mean Average Precision*) de 83,2% na base de dados 2007 PASCAL VOC e 82,2% na 2012 PASCAL VOC foram obtidos por W. Liu *et al.* [18]. A existência de redes profundas previamente treinadas evita que o treinamento se inicie com pesos completamente aleatórios, reduzindo assim o tempo para convergência da rede. Assim, sabendo que uma rede neural profunda é geralmente treinada para diferentes classificações, é possível ajustar as camadas para usá-la em outras aplicações, evitando assim dias, ou até mesmo semanas, de um treinamento que comece com pesos aleatórios [22].

2.4.1 *Transfer Learning* e *Fine-Tuning*

Transfer Learning é um método empregado na área de aprendizado de máquina que consiste em reutilizar um modelo desenvolvido para determinada tarefa como ponto de partida para solução de uma segunda tarefa [57].

Esse método costuma ser utilizado quando não é possível obter uma base de dados de larga escala anotada, como por exemplo classificar as opiniões sobre um dado produto, já que é necessário ler cada uma e definir se é positiva ou negativa. O modelo gerado para avaliar o impacto de um produto no mercado pode ser aplicado para outro produto, evitando uma nova anotação de base de dados e se aproveitando da similaridades de semântica nos textos de análise [58]. Assim, dois fatores são importantes na decisão da utilização da estratégia: o tamanho da nova base de dados e a similaridade entre as bases da tarefa para qual a rede foi inicialmente treinada com a segunda tarefa. Quando os pesos de um treinamento anterior são mantidos "congelados"(apenas para algumas determinadas camadas) e o treinamento é realizado novamente apenas para as camadas alteradas, utiliza-se o método de *Fine-Tuning*. Caso nenhum peso seja mantido fixo, teremos utilizado o *transfer learning*, e a rede foi treinada novamente com a mesma arquitetura [57].

Esse método se utiliza dos padrões aprendidos principalmente nas primeiras camadas de uma rede neural: linhas, contornos, cores. Assim, os pesos iniciais já contém informações relevantes para a definição da saída.

2.4.2 Ampliação de Dados (*Data Augmentation*)

Mesmo com uma alta similaridade, não recomenda-se realizar o *transfer learning* com bases de dados pequenas [22]. Uma maneira de se ampliar uma base de dados é empregando o método conhecido como Ampliação de Dados. Que ainda se divide em dois principais tipos:

- **Métodos Tradicionais**: O aumento do número de imagens na base de dados é feita utilizando-se transformações lineares: rotações, cortes, mudança de tamanho, perspectivas, cores dentre vários outros. Essas distorções geométricas foram utilizadas para aumentar o

conjunto de dados de treinamento em diferentes aplicações de redes neurais, a fim de se aumentar a acurácia do sistema [59] [60] .

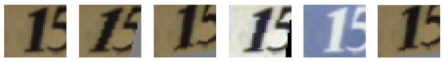


Figura 2.9: Imagens geradas por transformações geométricas (Modificado de [9]).

- **Redes Generativas Adversariais**: Um modelo desse tipo consiste em uma rede profunda formada por outras duas outras redes, colocadas uma contra a outra (daí o nome caracterizando a oposição), uma chamada de "geradora"é responsável por criar as imagens artificiais, a outra "distintiva"é responsável por avaliar a autenticidade das imagens criadas pela primeira. Tal método foi proposto por Ian Goodfellow *et al.* em 2014 [61].

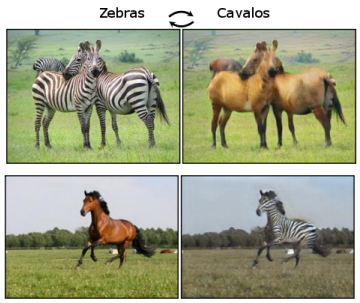


Figura 2.10: Imagens geradas por *GAN* (Modificado de [10]).

2.4.3 Sobreajuste e Subajuste

O Sobreajuste (ou *Overfitting*) ocorre quando o modelo apresenta resultados satisfatórios exclusivamente para o conjunto de dados utilizado no treinamento. Quando novos dados são apresentados, a rede não consegue prever as classes corretamente. Isso ocorre quando o modelo gerado é excessivamente complexo, já que o modelo está relacionando uma entrada x a sua saída y sem qualquer capacidade de generalização [62].

Enquanto o Subajuste (ou *Underfitting*) acontece quando mesmo no conjunto de dados de treinamento, o modelo não consegue classificar corretamente os dados. Ocorre por diversos motivos, dentre eles: modelo muito simples ou dados de entrada não contém *features* suficientemente expressivas para descrever uma classe. A generalização obtida é excessiva, a ponto de não ser possível perceber qualquer relação entre os dados e a saída desejada [62]. Ambos os ajustes são ilustrados na Figura 2.11.

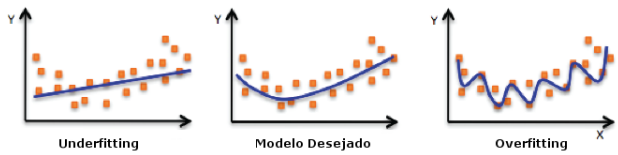


Figura 2.11: Tipos de ajustes à uma curva (Modificado de [11]).

Capítulo 3

Revisão da Literatura

Esse capítulo apresenta as técnicas empregadas em cada etapa de um sistema *ALPR*: localização da placa, segmentação e reconhecimento de caracteres. Essencialmente, os trabalhos realizados por Patel *et al.* [21], Du *et al.* [30] e Jørgensen [22] serviram como referência para o agrupamento e levantamento das técnicas já utilizadas, bem como uma análise qualitativa dos resultados obtidos, já que não seria apropriado comparar as acurácias de sistemas desenvolvidos e testados sob parâmetros e fatores variados [31]. Uma análise quantitativa é feita apenas com as arquiteturas de reconhecimento de objetos, na Seção 3.4, por possuir base de dados de larga escala utilizadas em competições.

3.1 Segmentação de Placa de Identificação Veicular numa Imagem

A primeira etapa consiste em encontrar e extrair uma placa de uma imagem, que por sua vez, pode conter uma, várias ou nenhuma placa veicular [22]. Sendo a etapa inicial, influencia significativamente as etapas subsequentes: se uma placa não foi localizada corretamente, as etapas de segmentação e reconhecimento de caracteres não são iniciadas. A placa pode estar ainda ocupando toda a imagem (considerando uma base de dados mais diversificada) ou uma parte menor da mesma, como a base utilizada neste trabalho [1].

3.1.1 Métodos Baseados em Bordas

Sabendo que as placas apresentam o mesmo formato (retangular para os carros brasileiros), é possível identificar placas numa imagem procurando por formas geométricas correspondentes na mesma, o que pode ser feito identificando fronteiras e arestas, como proposto em [63] e [64].

Sabendo que é possível detectar uma transição de cores entre o carro e a placa (caracterizando uma borda), o filtro de Sobel também já foi utilizado para realizar essa etapa em diversos trabalhos [65–68].

Sarfraz *et al.* [39] propôs que as regiões candidatas a possuírem uma placa fossem definidas a partir da coincidência de linhas de contornos verticais. Essa abordagem mostrou-se robusta ao se

analisar o problema da iluminação da captura: tal elemento não compromete as linhas da placa. Em [69], um algoritmo baseado também nas bordas verticais foi desenvolvido com a inteção de ser mais rápido que o operador de Sobel, e assim o fez, obtendo resultados de 7 a 9 vezes mais rápidos que o previamente citado.

A transformada de Hough também foi utilizada, capaz de detectar linhas verticais com até 30^{o} de inclinação, porém, o método apresentou um alto consumo de memória e tempo para chegar aos resultados [66].

A abordagem para identificar placas se baseando em bordas configura-se como uma das mais simples e rápidas, porém, ao se utilizar imagens mais complexas como entrada, ou mesmo imagens que apresentam muitas bordas, o desempenho costuma ser afetado negativamente [22].

3.1.2 Métodos Baseados em Textura

Esses métodos se baseiam na distribuição da intensidade dos pixels dentro da placa (a diferença entre os pixels que compõe os caracteres e aqueles que fazem parte do fundo da placa) [21].

Kaushik *et al.* [70] propôs um método que utilizava histogramas e janelas concêntricas na imagem identificasse mudanças súbitas na textura de uma potencial placa de veículo. Enquanto [71] sugeriu um método mais simples utilizando apenas as janelas concêntricas. Tais métodos contornam o problema anteriormente citado, já que não é necessário que a placa possua contornos bem definidos, porém, o custo computacional aumenta de maneira diretamente proporcional à complexidade da imagem, caso ela possua diferentes iluminações, por exemplo [22].

3.1.3 Métodos Baseados em Cores

Sabendo que a combinação das cores da placa e de seus caracteres é previamente conhecida (para cada país) e provavelmente só ocorre na região da placa [72], alguns métodos se baseiam nesse fundamento.

Em [73], o autor considera que as placas podem ser de apenas quatro diferentes cores e, assim, procura bordas que estejam na transição apenas entre essas cores. Enquanto [74] propôs que toda a imagem fosse analisada e caso algum pixel da cor de uma placa fosse localizado, que a cor dos pixels nas proximidades fosse analisada, e se dois ou mais pixels apresentarem essa cor, a região é então representada em uma nova imagem que contém todas as bordas que apresentaram essa característica.

Quando utilizados de maneira exclusiva, esses métodos estão fortemente sujeitos às variações de iluminação que podem existir. Logo, são utilizados em conjunto com outras estratégias [22].

3.1.4 Métodos Baseados na Localização de Caracteres

Esses métodos são definidos pela tentativa de localizar caracteres numa imagem, independente da sua localização, assim, uma placa é vista como uma sequência destes [22]. O trabalho desen-

volvido em [75] é um exemplo dessa abordagem, na qual uma rede neural classifica e enumera as regiões. Se várias regiões potencialmente possuem caracteres, a combinação linear dessas regiões é classificada como uma placa.

Enquanto [76] primeiramente identifica uma região com um caractere a partir da largura do mesmo e da diferença entre o fundo e a região do caractere, a informação da distância entre os caracteres também é utilizada, garantindo que uma placa seja extraída apenas se conter a quantidade de caracteres esperada.

Esses métodos apresentam bons resultados mesmo quando a imagem apresenta rotações, porém, caso uma imagem com variados textos seja utilizada, esses textos poderão ser classificados como placas [22].

Tabela 3.1: Comparação dos diferentes métodos de segmentação de placa. Modificado de [21].

Método	Vantagens	Desvantagens	Referências
Utilizando bordas	Simples e rápido.	Sensível a imagens complexas que apresentem muitas bordas.	[63–68]
Utilizando textura	Detecta uma placa mesmo com bordas deformadas.	Custo computacional alto para imagens com muitas bordas.	[70] e [71]
Utilizando cores	Detecta placas inclinadas e deformadas.	Condições de iluminação afetam negativamente o desempenho do sistema.	[72–74]
Utilizando localização dos caracteres	Robusto à rotações.	Indica diferentes textos de uma imagem como texto.Tempo de processamento maior.	[75] e [76]

3.2 Segmentação dos Caracteres

Nesta etapa, algumas placas segmentadas do processo da Seção 3.1 podem apresentar rotação ou iluminação não-uniforme, assim, um pré-processamento pode solucionar esses problemas. Em [38], o método de mínimos quadrados foi utilizado para contornar o problema da inclinação (tanto vertical quanto horizontal). Enquanto Lee *et al.* propôs a utilização de transformação bilinear [77]. Os métodos para segmentação dos caracteres foram agrupados em quatro grupos distintos, detalhados nas Subseções 3.2.1 a 3.2.3.

3.2.1 Métodos Baseados na Conectividade dos Pixels

Essa abordagem consiste em categorizar os pixels interconectados de uma imagem binarizada. Os pixels que possuem as mesmas propriedades dos caracteres (tamanho e proporção entre largura e altura) são considerados parte de um deles [21]. Os trabalhos desenvolvidos em [65, 78, 79] são exemplos de aplicações desse método, que se justifica por ser de simples implementação e apresenta bons resultados mesmo em imagens rotacionadas. Porém, se um caractere estiver cortado ou muito próximo de outro, o método começa a comprometer sua acurácia.

3.2.2 Métodos Baseados em Projeção de Perfis

Considerando que o caractere tem uma cor oposta à cor do fundo da placa, é possível (numa imagem binarizada) projetar verticalmente uma placa segmentada para se determinar o início e o fim da posição de cada caractere [21]. Em seguida, a região que potencialmente possui um caractere é projetada horizontalmente para que o mesmo seja reconhecido individualmente. Os trabalhos citados em [68] e [80] implementaram essa técnica. Sabendo que o método se baseia na projeção da imagem binarizada, o caractere pode estar em qualquer região da imagem e ainda assim ser localizado. Porém, é um método sensível a ruído, e é necessário conhecer previamente a quantidade de caracteres da placa.

3.2.3 Métodos Baseados no conhecimento prévio dos Caracteres

Dada uma imagem binarizada, é possível examiná-la por completo utilizando uma linha horizontal para encontrar os pontos de início e fim de um caractere: se a razão entre os pixels do fundo e os pixels do caractere excedem um dado limiar, considera-se que aquela linha delimita o início do caractere e o mesmo procedimento é válido para encontrar o fim do mesmo [67].

Outra estratégia é redimensionar a placa para um tamanho fixo, e partir de um modelo pré-definido, os caracteres são extraídos de regiões conhecidas como feito por Paliy *et al.* [81].

Em alguns casos especiais, como em Taiwan, todas as placas são da mesma cor, assim, se espera a mesma proporção de pixels brancos e pretos em toda as placas segmentadas, únicas cores presentes nas placas do país.

Tabela 3.2: Comparação entre diferentes métodos de segmentação de caracteres. Modificado de [21].

Método	Vantagens	Desvantagens	Referências
Utilizando conectividade dos pixels	Simples e robusto a rotações.	Não detecta caracteres que estejam distorcidos ou cortados.	[65, 78, 79]
Utilizando projeção de perfis	Resultado independe da posição do caractere e e robusto a rotações.	Ruídos interferem na projeção realizada e requer conhecimento prévio da quantidade de caracteres.	[68] e [80]
Utilizando conhecimento prévio de caracteres	Simples.	Limitado pelo conhecimento prévio, qualquer alteração pode ocasionar em erros.	[67] e [81]

3.3 Reconhecimento de Caracteres

O último passo consiste em reconhecer cada letra e número extraído da placa, comumente chamado de Reconhecimento Óptico de Caracteres (*Optical Character Recognition - OCR*) [82]. Assim, deve-se classificar uma imagem entre as 36 classes disponíveis (26 letras e 10 números).

3.3.1 Métodos Baseados em Correspondência de Modelos

Tais métodos se caracterizam por obter a similaridade de um caractere com um modelo pré-definido dele. O modelo (ou *template*) que mais se aproximar daquele caracter é definido como a previsão realizada [22]. Os trabalhos desenvolvidos em [83] e [84] utilizaram essa técnica em imagens binarizadas. Diferentes métricas de similaridade foram utilizadas: distância de Mahalanobis e regra de decisão de Bayes [78], índice de Jaccard [77], distância de Hausdorff [85] e distância de Hamming [39].

Esse método é simples, mas requer um modelo para cada caractere e cada fonte, além de considerar um tamanho fixo e não apresentar bons resultados quando imagens rotacionadas são utilizadas [22]. Seu processamento é consideravelmente longo pois compara um caractere com todos os modelos disponíveis.

3.3.2 Métodos Baseados em Aprendizado de Máquina

Ao utilizar esse tipo de método, é possível classificar um caractere se baseando em uma ou em múltiplas *features* [21]. Diferentes arquiteturas de redes neurais convolucionais apresentam bons resultados para essa tarefa. Li *et al.* [86] propôs uma rede pré-treinada com 9 camadas, equanto Jiao *et al.* [87] utilizou uma rede que aprendeu a classificar os caracteres se baseando na densidade da imagem.

Tabela 3.3: Comparação entre diferentes métodos de reconhecimento de caracteres. Modificado de [21].

Método	Vantagens	Desvantagens	Referências
Utilizando correspondência de modelos.	Simples.	Requer um modelo para cada caractere de cada fonte e não apresenta bons resultados com caracteres rotacionados ou danificados.	[39, 77, 78, 83–85].
Utilizando aprendizado de máquina.	Extrai múltiplas *features* de cada caractere além de ser robusto à distorções.	A extração de *features* demanda muito tempo, e caso não seja feita adequadamente, pode degradar o reconhecimento.	[86] e [87]

3.4 Redes Neurais para Reconhecimento de Objetos

Uma rede neural desenvolvida para reconhecer um objeto deve ser capaz de detectar, segmentar e classificá-lo [22]. A competição PASCAL VOC [88] realizada anualmente de 2005 até 2012 foi criada para comparar o desempenho de redes neurais desenvolvidas para realizar tal tarefa, utilizando-se uma base de dados de larga escala, com mais de 1000 classes de objetos.

No trabalho realizado por Zhao *et al.* em 2018 [89], os *frameworks* mais utilizados foram divididos entre os que são baseados em regiões de interesse e os que se baseiam em regressão.

3.4.1 *Frameworks* baseados em Regiões de Interesse

As abordagens dessa subseção são inspiradas no processo de reconhecimento humano: inicialmente, todo o cenário é visto e posteriormente apenas a região de interesse é analisada [89].

3.4.1.1 *Region-Convolutional Neural Network* (*R-CNN*)

A rede proposta em 2014 por Ross Girshick [12] alcançou um *Mean Average Precision* (*mAP*) de 53,3% na base de dados PASCAL VOC 2012 e pode ser dividida em três módulos.

No primeiro módulo, 2 mil regiões são definidas na imagem de entrada, utilizando o método de busca seletiva proposto em [90]. Em seguida, essas regiões são deformadas para adquirir o formato de um quadrado e funcionar como a entrada de uma rede neural [43] que produz um vetor de *features* com dimensão 4096 como saída. Assim, cada região definida possui uma representação de *features* robusta e com alto nível de abstração, dada a alta capacidade de aprendizado das redes neurais [89]. Por fim, o último módulo utiliza a representação de *features* obtida anteriormente numa Máquina de Vetores de Suporte (em inglês, *Support Vector Machine - SVM*) linear, classificando cada região em cada uma das categorias possíveis. A Figura 3.1 representa todos os módulos de uma rede neural desse tipo.

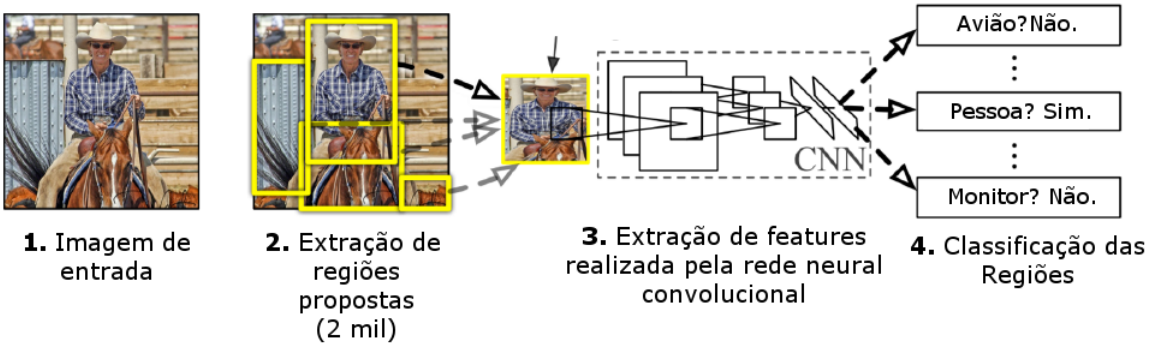


Figura 3.1: Visão geral de uma *R-CNN*. Modificado de [12].

3.4.1.2 *Spatial Pyramid Pooling-net* (*SPP-net*)

Uma das limitações das redes neurais é o requisito de que as entradas das camadas totalmente conectadas (*Fully Connected-FC*) precisam ter exatamente o mesmo tamanho e comprimento [13]. Assim, é necessário redimensionar ou cortar a imagem entre as camadas, o que pode causar distorções geométricas indesejadas, ou mesmo perder partes relevantes do cenário, como ilustrado na Figura 3.2. He *el at.* [13] desenvolveu uma estratégia para contornar essa limitação das redes neurais convolucionais, que consiste em adicionar uma camada *Spatial Pyramid Pooling* (*SPP*) após a última camada convolucional. O método é um dos mais bem sucedidos em visão computacional [91] e consiste em dividir a imagem em seções que possuem diferentes níveis de complexidade e *features* locais associados a elas. Assim, se diferenciando da *R-CNN*, a arquitetura *SPP-net* reutiliza o mapa de *features* da quinta camada de convolução para projetar regiões de interesse

de tamanhos arbitrários para um vetor de *features* de tamanho fixo. A viabilidade de se realizar essa consideração se deve ao fato do mapa não representar apenas os pesos das respostas daquela camada, mas ter relação com sua posição espacial na rede, como afirmado em [13].

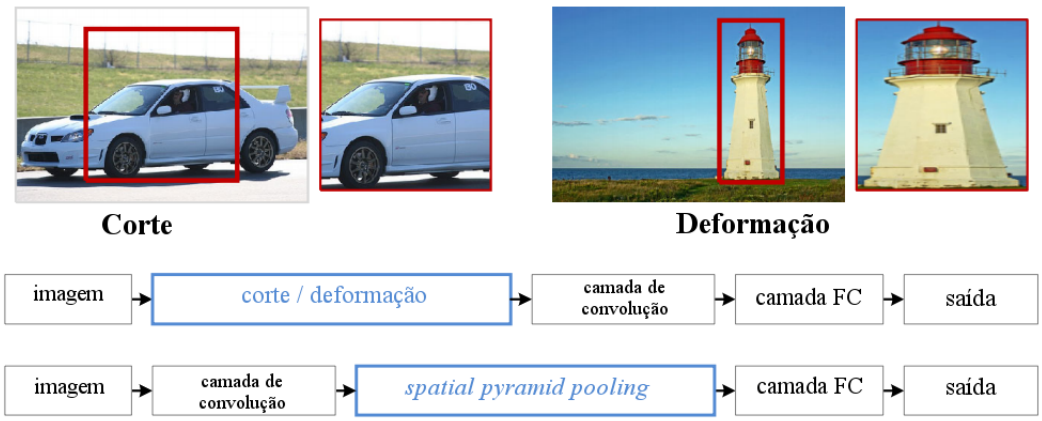


Figura 3.2: Parte superior: corte e deformação. Parte mediana: *CNN* Convencional. Parte inferior: Abordagem *SPP-net*. Modificado de [13].

3.4.1.3 *Fast Region-Convolution Neural Network* (*Fast R-CNN*)

Uma das principais desvantagens da *R-CNN* é a necessidade de percorrer todas as camadas de uma rede neural convolucional para cada região proposta. Consequentemente, quando há interseção de alguma área entre duas regiões propostas, essa área é processada múltiplas vezes [22].

Assim, a rede proposta por Girshick em 2015 [92] utiliza o conceito de *Region of Interest Pooling* (*RoIPool*), que consiste num caso especial de camada *SPP* com um nível. Existe apenas um mapa compartilhado de *features* para toda a imagem de entrada, e para cada objeto de interesse, um vetor de *features* de tamanho fixo é extraído do mapa global. Em seguida, cada vetor de features é passado a camadas completamente conectadas até que as duas camadas de saída são alcançadas. Uma dessas camadas é responsável por indicar a probabilidade para todas as classes de objetos, enquanto a outra retorna os valores das coordenadas que delimitam a região estimada.

Por fim, uma camada de regressão é adicionada e o classificador *SVM* é substituído por uma camada *softmax*. Uma limitação da *R-CNN* consistia em não ser possível atualizar as camadas da rede durante o treinamento (por apresentar três módulo distintos), porém isso é possível na *Fast R-CNN* que apresenta apenas um módulo.

3.4.1.4 *Faster Region-Convolutional Neural Network* (*Faster R-CNN*)

Ren *et al.* em 2015 [93] introduziu o conceito de Rede de Regiões Propostas (*Region Proposal Network - RPN*), para substituir os métodos tradicionais de definição das regiões isoladas . Uma rede *RPN* é formada exclusivamente de camadas convolucionais, que recebe como entrada uma imagem e gera como saída regiões retangulares. Assim, a implementação de uma rede *Faster R-*

CNN consiste em percorrer o mapa de *features* com uma janela de tamanho fixo n x n pixels. Para cada ponto, *k anchor boxes* com 4 coordenadas são sugeridas (*bounding boxes* que funcionam como referência inicial para o reconhecimento do objeto), bem como dois valores de confiabilidade de cada *anchor box*: um para a classificação do objeto e outro para as coordenadas definidas.

3.4.1.5 *Feature Pyramid Network* (*FPN*)

Pirâmides de *features* tem sido utilizadas em redes neurais de reconhecimento de objetos por apresentarem invariância à mudança de escala [94]. Porém, apresentam como desvantagem um rápido crescimento do consumo de tempo e memória com o avanço do treinamento. A Figura 3.3 mostra o porquê da arquitetura *FPN* ter causado significante melhorias de acurácia, funcionando como um extrator de *features* genéricas em diversos trabalhos [14].

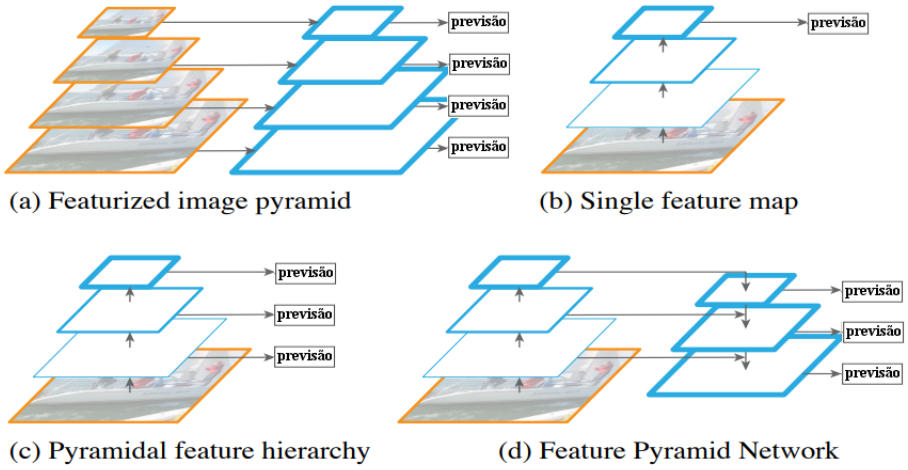


Figura 3.3: Diferentes estruturas piramidais. a) Consiste num método lento, já que a pirâmide de *features* é criada a partir da pirâmide de imagens. b) Apenas uma escala de *features* é utilizada para reduzir o tempo computacional. c) Uma alternativa à subfigura (a) é utilizar a hierarquia piramidal de *features* já definidas por uma rede convolucional. d) Uma rede *FPN* incorpora (b) e (c). Contornos azuis indicam mapas de *features* e linhas com maior espessura representam *features* com valor semântico mais significativo. Modificado de [14].

3.4.1.6 *Mask Region-Convolutional Neural Network* (*Mask R-CNN*)

Considerado o estado da arte na área de segmentação de imagens, a rede *Mask R-CNN* proposta por He *et al.* em [15] é capaz de segmentar uma imagem até mesmo no nível dos pixels e pode ser considerada como uma modificação da *Faster R-CNN*, pois acrescenta uma funcionalidade (calcula a máscara de um objeto), além de calcular a *bounding box* do mesmo e realizar a classificação.

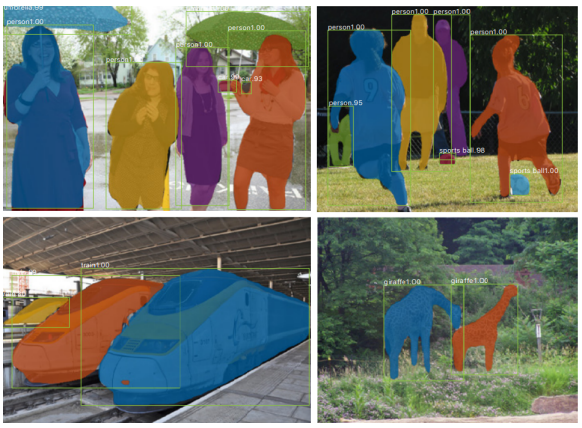


Figura 3.4: Exemplos de segmentações realizadas pela *Mask R-CNN* [15].

Uma das maiores contribuições feitas por esse modelo foi o refinamento das **Regiões de Interesse** (*Regions of Interest - RoI*) pelo método definido como *RoIAlign*. Extrair pequenos mapas de *features* das regiões de interesse é uma operação comum, denominada *RoIPool* [92], porém quando o mapeamento de uma região de interesse é realizado para o mapa de *features*, o mesmo não ocorre de maneira precisa. Assim, considerando que a segmentação de imagens demanda o melhor ajuste possível, o alinhamento utilizado pela *Mask R-CNN* utiliza interpolação bilinear, que consiste em calcular uma média ponderada pela distância de quatro pixels quando uma região de interesse não pode ser exatamente definida no mapa de *features* (como ilustrado na Figura 3.5), fazendo com que a acurácia do sistema aumente significativamente [15].

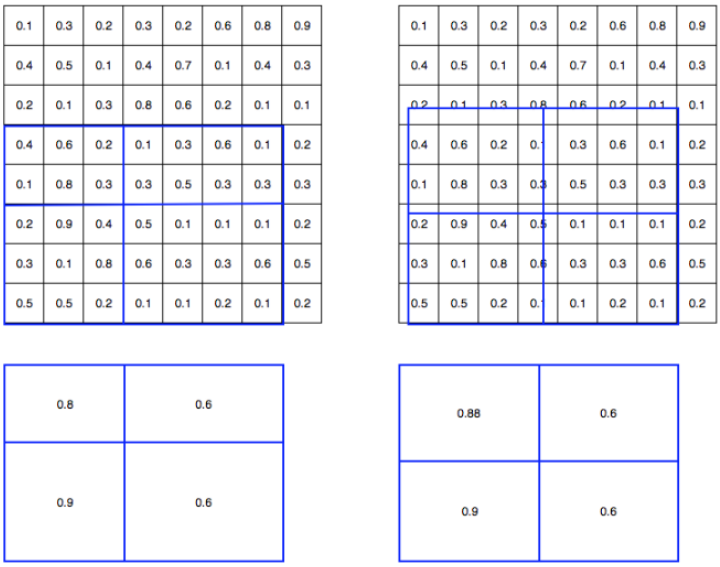


Figura 3.5: Mapeamento realizado pelo *Faster R-CNN* (à esquerda) e mapeamento realizado pelo *Mask R-CNN* (à direita) [16].

3.4.2 *Frameworks* baseados em Regressão

Para aplicações que possuem o requisito de tempo real, o tempo de execução das várias etapas dos *frameworks* baseados em seleção de regiões de interesse se torna um gargalo da estratégia como afirmado por Zhao *et al.* [89] que também observou que sistemas que possuem apenas uma etapa baseada em regressão global mapeiam diretamente os pixels para as *bounding boxes*, reduzindo o tempo levado para detecção.

3.4.2.1 *You Only Look Once* (*YOLO*)

A primeira arquitetura de reconhecimento de objetos a obter sucesso na detecção utilizando apenas uma etapa foi a proposta por Redmon *et al.* em 2015 [17], que não possuía qualquer etapa de gerar regiões propostas, apenas a imagem sendo processada uma vez por apenas uma rede neural convolucional resultando numa detecção até 6-7 vezes mais rápida que uma rede *Faster R-CNN*, com um tempo de detecção de 22 ms [22]. Logo, tal arquitetura é capaz de processar vídeos em tempo real, porém a desvantagem apresentada é uma perda na acurácia. O *YOLO* atinge um mAP de 63.4% na base de dados VOC 2007, o que é inferior às arquiteturas *R-CNN*.

No primeiro passo, a imagem de entrada é dividida em uma grade de S x S pixels e cada célula da grade é responsável por detectar o objeto que está localizado em seu centro. Em seguida, uma célula deve informar uma quantidade B de *bounding boxes* e seus respectivos níveis de confiança, calculados formalmente como $Pr(Object) * IOU_{pred}^{truth}$.

Na qual $Pr(Object)$ representa a probabilidade de um objeto existir na região delimitada pela *bounding box* (BB) definida e IOU_{pred}^{truth} representa a acurácia da previsão realizada para a BB, utilizando a interseção sobre união (IoU) das coordenadas previstas com as coordenadas do *Ground Truth* (GT) [17]. Simultânea e independentemente do número de classes, C probabilidades condicionais de cada classe também são estimadas ($Pr(Class_i|Object)$) para cada célula da grade que possui um objeto, como ilustrado na Figura 3.6.

Por fim, a Equação 3.1 define como a probabilidade de uma classe aparecer numa BB e o quão bem as coordenadas definidas contornam o objeto, na qual a probabilidade da classe é multiplicada pelo nível de confiança de cada BB, chegando assim à confiança de cada classe em cada BB.

$$Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth}. \tag{3.1}$$

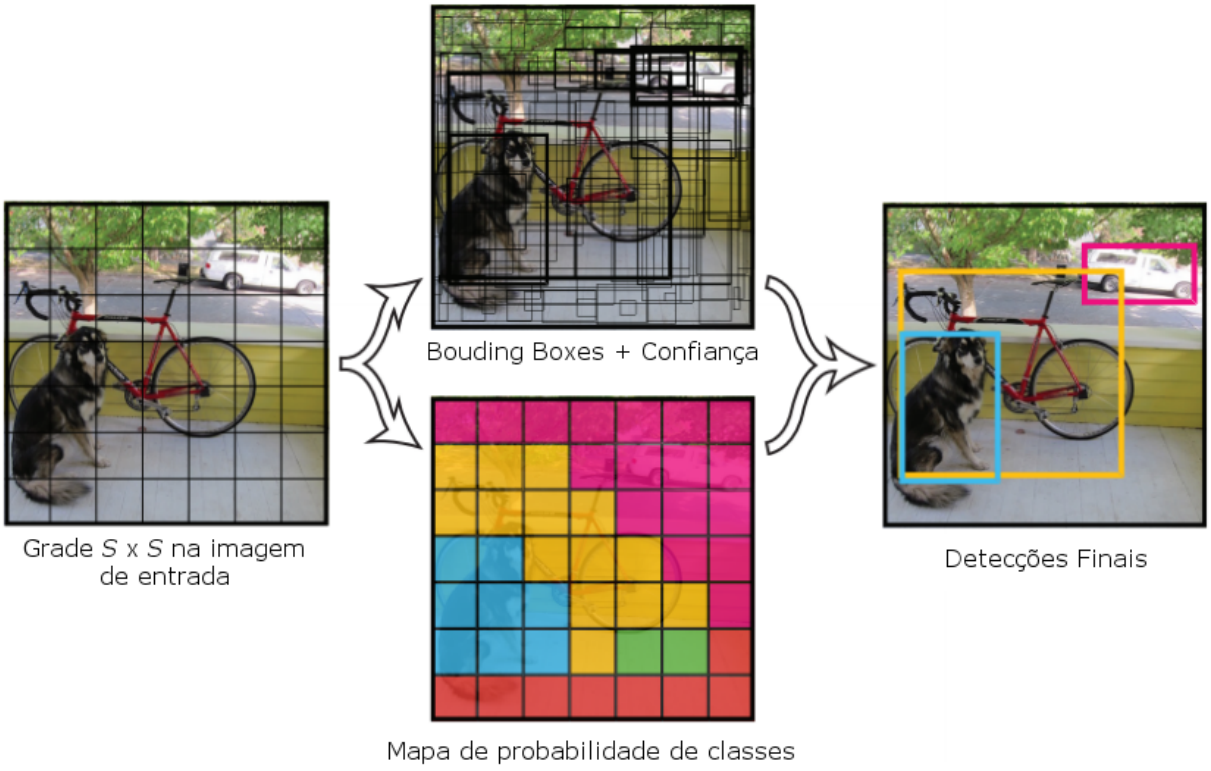


Figura 3.6: Visão geral do *YOLO*. A imagem dividida em células tem suas BB previstas, confiança para essas caixas e C probabilidades de cada classe. Modificado de [17].

A função de perda utilizada nessa arquitetura é formalmente definida pela Equação 3.2 que é otimizada durante o treinamento.

$$\begin{aligned}
&\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\
&+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2.
\end{aligned} \tag{3.2}$$

Na qual uma célula i (x_i,y_i) representa o centro da BB relativa às bordas da célula (w_i,h_i) e normalizados em relação ao tamanho da imagem. C_i representa o nível de confiança [17]. Para enfatizar a acurácia das coodernadas da BB prevista, o termo λ_{obj} é adicionado (por padrão é igual a 5), enquanto o termo λ_{noobj} é adicionado para que as BB's que não possuam objetos e consistam apenas no fundo da imagem não interfiram majoritariamente na função de perda (por padrão, $\lambda_{noobj} = 0.5$), evitando um problema de desequilíbrio nas classes.

Analisando mais detalhadamente, o primeiro termo da soma penaliza a função caso os centróides definidos pela previsão sejam diferentes dos centróides esperados. O segundo termo penaliza os casos em que o tamanho da BB prevista está incorreto.

Tanto o terceiro quanto o quanto o quarto termo da equação se referem, respectivamente, à perda do nível de confiança para os casos em que existe um objeto e àqueles onde não existe um objeto, onde $\hat{C}_i$ representa o nível de confiança da BB j na célula i.

O último termo caracteriza a perda na classificação, que é igual ao erro quadrático da probabilidade condicional $\hat{p}_i(c)$ de cada classe c para cada célula i.

Analisando a função de perda utilizada pelo *YOLO*, nota-se que a rede resolve um problema de regressão e não de classificação como as *R-CNNs* A arquitetura dessa rede foi inspirada pela *GoogleNet* e é formada por 24 camadas convolucionais, seguidas de 2 camadas completamente conectadas. A versão *Fast Yolo* utiliza apenas 9 camadas convolucionais, reduzindo significativamente o tempo de detecção, porém comprometendo a acurácia [22].

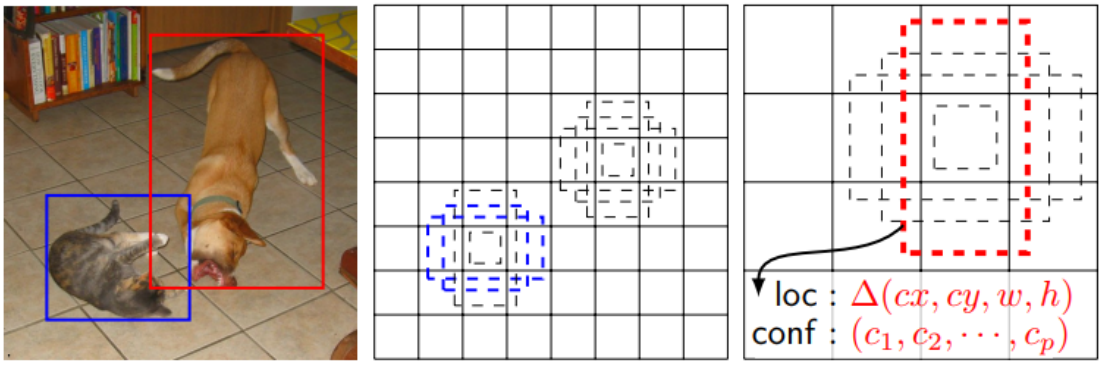
Em 2016, uma nova versão do *YOLO* foi proposta [46] com diversas melhorias em relação à primeira versão: normalização de grupos (*batch normalization*), melhorando a taxa de convergência e evitando o sobreajuste (aumento de 2% no *mAP* [22]). Aumento da resolução do classificador de 224 x 224 para 448 x 448, ajustando melhor os resultados (aumento de 4% no *mAP* [22]), porém afetando negativamente o tempo de treinamento.

Para que o modelo comece com melhores representações das BBs, a arquitetura *YOLO* v2 utiliza agrupamento por k-médias. Para aumentar a acurácia em imagens de diferentes tamanhos, caracterizando o treinamento multiescala. Durante esse treinamento o tamanho das imagens não é mantido fixo, mas uma resolução nova é utilizada a cada *batches*, variando de 320 x 320 pixels até 608 x 608 pixels. Fazendo com que a rede mantenha bons resultados com variações de tamanhos das imagens. [22].

3.4.2.2 *Single Shot Multibox Detector* (*SSD*)

Uma desvantagem do *YOLO* é não apresentar bons resultados ao detectar objetos pequenos, pois as coodernadas previstas para as BBs impõem restrições espaciais, como concluído pelos próprios autores [17]. É importante ressaltar também que as operações realizadas pelo treinamento multiescala apresentam bons resultados com imagens de diferente tamanhos, mas não em imagens com proporções variadas, produzindo *features* relativamente grosseiras que não são capazes de classificar corretamente os objetos [89].

A fim de desenvolver uma rede que não apresentasse esses problemas, Li *et al.* [18] propôs o *Single Shot Multibox Detector* (*SSD*), que diferentemente do *YOLO*, não utiliza grades de tamanhos fixos, mas que utiliza um conjunto padrão de BBs de referência com diferentes proporções e tamanhos [18]. Tal estratégia está exemplificada na Figura 3.7



(a) *Bounding Boxes* do *Ground Truth* (b) Mapa de *features* 8 x 8 (c) Mapa de *features* 4 x 4

Figura 3.7: Visão geral do método *SDD*. (a) Para o treinamento, a rede recebe uma imagem de entrada com a definição da localização dos objetos (GT). Para cada BB de referência, além das coordenadas, o nível de confiança de cada classe também é calculado $(c_1, c_2, ..., c_p)$. Durante o treinamento, cada BB de referência é comparada com a BB do GT. Nos mapas de diferentes escalas desse exemplo, 2 BBs de referência coincidiram e foram selecionadas para o gato (b) enquanto uma foi selecionada para o cachorro (c). Modificado de [18].

Capítulo 4

Metodologia

A metodologia apresentada neste capítulo é descrita pelos fluxogramas das Figuras 4.1 e 4.2. As Subseções apresentarão cada uma das etapas da metodologia proposta em seus detalhes.

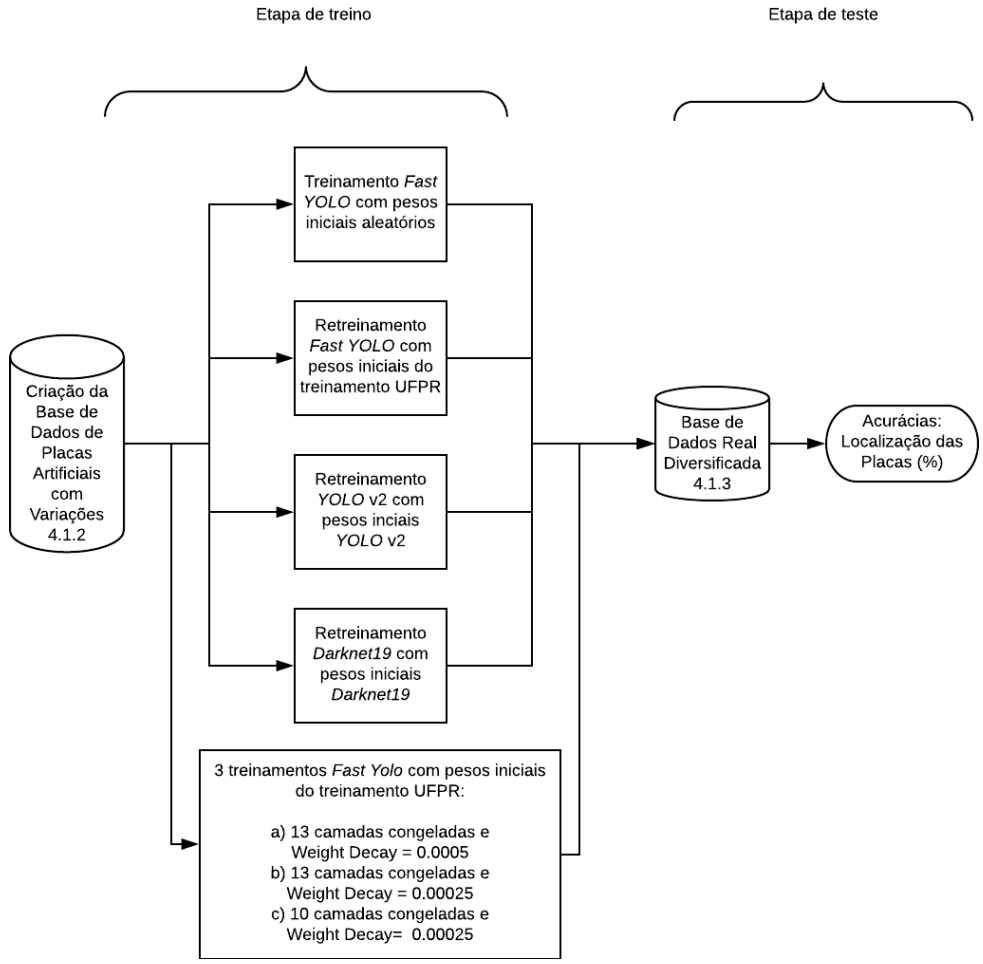


Figura 4.1: Metodologia para os treinamentos e testes das arquiteturas de localização da placa.

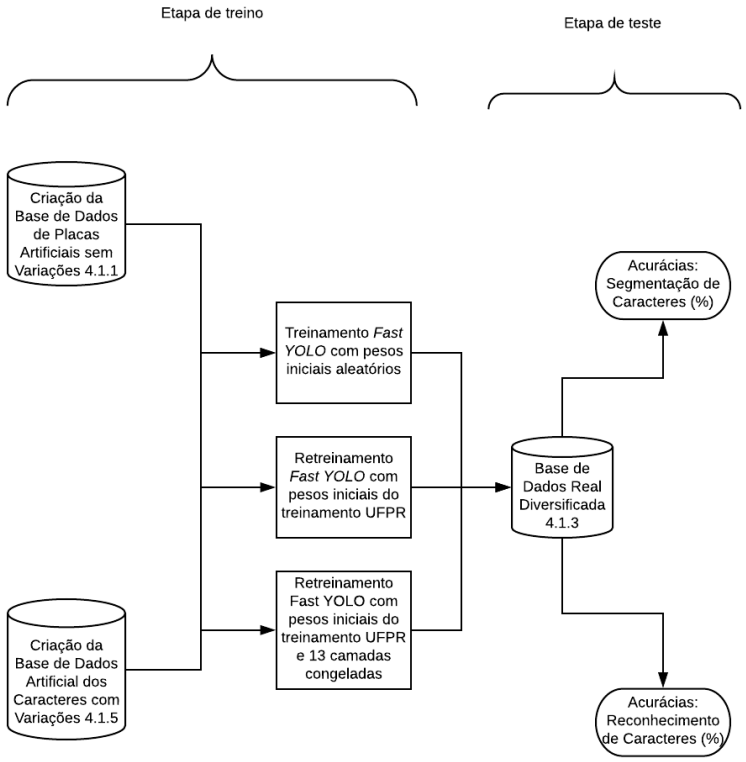


Figura 4.2: Metodologia para os treinamentos e testes das arquiteturas de segmentação e reconhecimento dos caracteres.

4.1 Bases de Dados

4.1.1 Base Artificial sem Variações

Essa base de dados (representada na Figura 4.4) consiste em placas criadas utilizando a biblioteca *OpenCV* [95], apenas com os 7 caracteres que identificam uma placa. O nome da cidade e do estado foram omitidos para evitar o sobreajuste, evitando que a rede neural extraísse dessa parte da placa *features* que não contribuiriam para a utilização com placas de diferentes locais.

A fonte utilizada foi a *Mandatory* conforme especificado na resolução do Contran [45]. As cores e um gradiente de cinza do fundo da placa também foram definidos de forma a aproximar-se de placas reais. Foram criadas 902 placas artificiais nessa base de dados, tal quantidade se justifica pelo número de sequências únicas de caracteres da base de dados real utilizada na etapa de teste e ilustrada na Subseção 4.1.3.

Justifica-se a utilização dessa base de dados por ser de fácil anotação dos caracteres utilizados bem como a posição de cada caractere (para a rede neural responsável por segmentá-los).

Figura 4.3: Base de Dados Artificial sem Variações.

4.1.2 Base Artificial com Variações

Para que uma base de dados mais complexa fosse obtida, a base de dados *SUN397*, disponibilizada pelo *Massachusetts Institute of Technology* [96] foi utilizada como imagem de fundo, na qual uma placa foi adicionada com variações aleatórias e limitadas de rotação, escala, ruído Gaussiano, brilho, perspectiva e aguçamento. Tais variações foram obtidas utilizando a ferramenta *Data Augmentation for Object Detection(YOLO)* [97]. Assim, as imagens já tiveram suas anotações criadas no formato aceito pelo *framework* utilizado no momento da sua criação.

Figura 4.4: Base de Dados Artificial com Variações.

4.1.3 Base de Dados Real Diversificada

Para que se tivesse um Base de Dados que não foi utilizada em nenhum treinamento citado nesse trabalho, as imagens disponíveis em [1] foram utilizadas como referência para todos os cálculos de

acurácia realizados e apresentados com a intenção de analisar os diferentes parâmetros no *transfer learning* no Capítulo 5.

Justifica-se tal escolha por ser uma base de dados com placas reais, capturadas em diferentes ambientes sob circunstâncias variadas. Assim, as placas dessa base de dados ocupam diferentes proporções da imagem, distorções e até mesmo desgaste natural, como ilustrado na Figura 4.5.

Tal conjunto de dados apresenta 1126 imagens capturadas, sendo que considerando sequências de caracteres únicos, apresenta 902 placas diferentes. As imagens possuíam, inicialmente, apenas anotação sobre quais eram os caracteres que a formavam. Assim, foi necessário ainda definir as *BBs* para o *Ground Truth* da base.

Figura 4.5: Base de Dados Real Diversificada.

4.1.4 Base de Dados Real da Universidade Federal do Paraná (UFPR-ALPR)

Dada a dificuldade em se obter base de dados com placas de carros brasileiras já anotadas, a base utilizada e desenvolvida por Laroca *et al.* [19] foi utilizada com algumas modificações.

A base possui 4500 imagens de 150 veículos diferentes (cada uma das 3 câmeras utilizadas capturaram 1500 imagens). Como o trabalho para o qual foi desenvolvida utiliza informações de redundância de *frames*, ele não se aplica aqui. Assim, apenas o primeiro frame de cada sequência foi utilizado, reduzindo a base utilizada a 150 imagens. Exemplos de imagens presentes na base estão ilustrados na Figura 4.6, na qual nota-se que a distância entre a câmera e a placa não apresenta grandes variações.

Figura 4.6: Exemplos de imagens da base UFPR-ALPR [19].

4.1.5 Bases Artificiais para o treinamento do Reconhecimento de Caracteres

Para realizar o retreinamento das redes de reconhecimento de letras e números, bases artificiais foram criadas para cada grupo utilizando a ferramenta *Data Augmentation for Object Detection (YOLO)* [97]. Sendo que 50 variações de cada dígito foram utilizadas (totalizando 500 imagens), e 20 variações de cada letra (totalizando 520 imagens). A Figura 4.7 mostra exemplos de imagens dessas bases de dados.

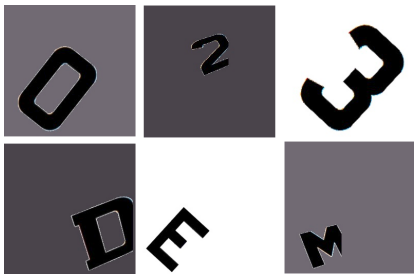


Figura 4.7: Base de Dados Real Artificial para Reconhecimento de Caracteres

4.2 Arquiteturas de Redes Neurais Utilizadas

Todas as arquiteturas descritas nessa Seção foram implementadas utilizando-se o *framework* de redes neurais *darknet* [98], escrito em linguagem C e que apresenta otimizações para a execução em *GPUs*. A escolha da ferramenta justifica-se pelo fato do único trabalho brasileiro para a tarefa de reconhecimento de placas disponibilizado ter sido desenvolvido nessa plataforma [19]. Logo, a compatibilidade foi mantida com os arquivos de treinamento gerados anteriormente. Ademais, a arquitetura *YOLO* e suas variações utilizadas nesse trabalho também foram criadas nesse *framework*.

Nesta seção, o número de filtros da última camada de todas as arquiteturas é calculado segundo a fórmula do *YOLO*:

$$filtros = (C + 5) * A \tag{4.1}$$

Na qual C representa o número de classes, e A os *anchor boxes* que no caso sempre será igual a 5: quatro coordenadas da BB e a confiança da previsão realizada. Os parâmetros fixos para os treinamentos foram: $momentum = 0.9$ e taxa de aprendizado igual a 0.001, 0.0001 e 0.00001 até as iterações 100, 25000 e 35000, respectivamente. A proporção utilizada para o conjunto de dados de treinamento e teste foi de 75:25% como feito em [99].

4.2.1 Localização da Placa na Imagem (*Fast YOLO*)

Considerando que o estudo da análise do treinamento de uma rede neural com base de dados sintética envolve vários treinamentos com variações de parâmetros, optou-se por utilizar a *Fast YOLO* por apresentar um menor número de camadas, e por consequência, um menor tempo de convergência (para o treinamento) e detecção (para o teste). Assim, dois parâmetros foram analisados (em 3 treinamentos) com o intuito de se observar a influência dos mesmos na acurácia da rede: i)Congelamento de 13 ou 10 camadas e ii) Decaimento de Pesos igual a 0.0005 ou 0.00025.

Tal arquitetura reduz a acurácia (na base de dados VOC 2007, houve uma redução no *mAP* de 10.4%), mas é capaz de reconhecer objetos em imagens amostradas a uma taxa de 155 frames por segundo [22].

Tabela 4.1: Rede Utilizada para Localização da Placa (*Fast YOLO* - [19]).

	Camadas	**Filtros**	**Tamanho**	**Entrada**	**Saída**
0	conv	16	3 x 3/1	416 x 416 x 3	416 x 416 x 16
1	max		2 x 2/1	416 x 416 x 16	208 x 208 x 16
2	conv	32	3 x 3/1	208 x 208 x16	208 x 208 x 32
3	max		2 x 2/2	208 x 208 x 32	104 x 104 x 32
4	conv	64	3 x 3/1	104 x 104 x 32	104 x 104 x 64
5	max		2 x 2/2	104 x 104 x 64	52 x 52 x 64
6	conv	128	3 x 3/1	52 x 52 x 64	52 x 52 x 128
7	max		2 x 2/2	52 x 52 x 128	26 x 26 x 128
8	conv	256	3 x 3/1	26 x 26 x 128	26 x 26 x 256
9	max		2 x 2/1	26 x 26 x 256	13 x 13 x 256
10	conv	512	3 x 3/1	13 x 13 x 256	13 x 13 x 512
11	max		2 x 2/1	13 x 13 x 512	13 x 13 x 512
12	conv	1024	3 x 3/1	13 x 13 x 512	13 x 13 x 1024
13	conv	1024	3 x 3/1	13 x 13 x 1024	13 x 13 x1024
14	conv	30	1 x 1/1	13 x 13 x 1024	13 x 13 x 30
15	detecção				

4.2.2 Localização da Placa na Imagem (*YOLO* v2)

Para que outra arquitetura fosse testada na detecção de placas numa imagem, a YOLO (inspirada na *Darknet-19* [46]) foi utilizada.

Tabela 4.2: Rede utilizada para localização da placa (*YOLO* v2 [22])

	Camadas	**Filtros**	**Tamanho**	**Entrada**	**Saída**
1	conv	32	3 x 3/1	416 x 416 x 3	416 x 416 x 32
2	max		2 x 2/2	416 x 416 x 32	208 x 208 x 32
3	conv	64	3 x 3/1	208 x 208 x 32	208 x 208 x 64
4	max		2 x 2/2	208 x 208 x 64	104 x 104 x 64
5	conv	128	3 x 3/1	104 x 104 x 64	104 x 104 x 128
6	conv	64	1 x 1/1	104 x 104 x 128	104 x 104 x 64
7	conv	128	3 x 3/1	104 x 104 x 64	104 x 104 x 128
8	max		2 x 2/2	104 x 104 x 128	52 x 52 x 128
9	conv	256	3 x 3/1	52 x 52 x 128	52 x 52 x 256
10	conv	128	1 x 1/1	52 x 52 x 256	52 x 52 x 128
11	conv	256	3 x 3/1	52 x 52 x 128	52 x 52 x 256
12	max		2 x 2/2	52 x 52 x 256	26 x 26 x 256
13	conv	512	3 x 3/1	26 x 26 x 256	26 x 26 x 512
14	conv	256	1 x 1/1	26 x 26 x 512	26 x 26 x 256
15	conv	512	3 x 3/1	26 x 26 x 256	26 x 26 x 512
16	conv	256	1 x 1/1	26 x 26 x 512	26 x 26 x 256
17	conv	512	3 x 3/1	26 x 26 x 256	26 x 26 x 512
18	max		2 x 2/2	26 x 26 x 512	13 x 13 x 512
19	conv	1024	3 x 3/1	13 x 13 x 512	13 x 13 x 1024
20	conv	512	1 x 1/1	13 x 13 x 1024	13 x 13 x 512
21	conv	1024	3 x 3/1	13 x 13 x 512	13 x 13 x 1024
22	conv	512	1 x 1/1	13 x 13 x 1024	13 x 13 x 512
23	conv	1024	3 x 3/1	13 x 13 x 512	13 x 13 x 1024
24	conv	1024	3 x 3/1	13 x 13 x 1024	13 x 13 x 1024
25	conv	1024	3 x 3/1	13 x 13 x 1024	13 x 13 x 1024
26	route 17				
27	conv	64	1 x 1/1	26 x 26 x 512	26 x 26 x 64
28	reorganize		/2	26 x 26 x 64	13 x 13 x 256
29	route 28 25				
30	conv	1024	3 x 3/1	13 x 13 x 1280	13 x 13 x 1024
31	conv	30	1 x 1/1	13 x 13 x 1024	13 x 13 x 30
32	detecção				

4.2.3 Localização da Placa na Imagem (Modelo *Darknet19*)

Desenvolvida para cumprir o desafio *ImageNet* [100] de 1000 classes, a arquitetura da Tabela 4.3 também foi utilizada para o reconhecimento de placas veiculares.

Tabela 4.3: Rede utilizada para localização da placa na imagem (Modelo *Darknet19* [23]

	Camadas	**Filtros**	**Tamanho**	**Entrada**	**Saída**
1	conv	32	3 x 3/1	256 x 256 x 3	256 x 256 x 32
2	max		2 x 2/2	256 x 256 x 32	128 x 128 x 32
3	conv	64	3 x 3/1	128 x 128 x 32	128 x 128 x 64
4	max		2 x 2/2	128 x 128 x 64	64 x 64 x 64
5	conv	128	3 x 3/1	64 x 64 x 64	64 x 64 x 128
6	conv	64	1 x 1/1	64 x 64 x 128	64 x 64 x 64
7	conv	128	3 x 3/1	64 x 64 x 64	64 x 64 x 128
8	max		2 x 2/2	64 x 64 x 128	32 x 32 x 128
9	conv	256	3 x 3/1	32 x 32 x 128	32 x 32 x 256
10	conv	128	1 x 1/1	32 x 32 x 256	32 x 32 x 128
11	conv	256	3 x 3/1	32 x 32 x 128	32 x 32 x 256
12	max		2 x 2/2	32 x 32 x 256	16 x 16 x 256
13	conv	512	3 x 3/1	16 x 16 x 256	16 x 16 x 512
14	conv	256	1 x 1/1	16 x 16 x 512	16 x 16 x 256
15	conv	512	3 x 3/1	16 x 16 x 256	16 x 16 x 512
16	conv	256	1 x 1/1	16 x 16 x 512	16 x 16 x 256
17	conv	512	3 x 3/1	16 x 16 x 256	16 x 16 x 512
18	max		2 x /2/2	16 x 16 x 512	8 x 8 x 512
19	conv	1024	3 x 3/1	8 x 8 x 512	8 x 8 x 1024
20	conv	512	1 x 1/1	8 x 8 x 1024	8 x 8 x 512
21	conv	1024	3 x 3/1	8 x 8 x 512	8 x 8 x 1024
22	conv	512	1 x 1/1	8 x 8 x 1024	8 x 8 x 512
23	conv	1024	3 x 3/1	8 x 8 x 512	8 x 8 x 1024
24	conv	30	1 x 1/1	8 x 8 x 1024	8 x 8 x 30
25	detecção				

4.2.4 Segmentação de Caracteres (*YOLO-VOC* Modificado)

Como verificado por Gonçalves *et al.* [101], a separação das redes neurais responsáveis por identificar os números e as letras reduz as classificações incorretas. O tamanho de entrada da imagem da rede responsável por segmentar os caracteres (240 x 80) se justifica pela proporção entre as dimensões das placas de carro brasileiras (3:1). O trabalho que propôs essa arquitetura [24] testou também as dimensões de entrada: 144 x 48 → 18 x 6 e 192 x 64 → 24 x 8, porém as redes com essas entradas apresentaram desempenho inferior.

É importante ressaltar que os caracteres segmentados são ordenados de forma decrescente de acordo com sua coordenada no eixo horizontal, separando assim os 3 primeiros que são letras, dos 4 últimos que são os números, evitando assim classificações incorretas dos pares: O/Q, 0/D, 1/I, 5/S, 2/Z, B/8, A/4 [24].

Montazzolli *et al.* percebeu que a granularidade da arquitetura original do *YOLO* (13 x 13) não seria suficiente para identificar os 7 caracteres, assim, propôs que a saída final da rede tivesse dimensões 30 x 10, quase triplicando a granularidade horizontal [24].

Tabela 4.4: Rede utilizada para segmentação dos caracteres, utilizando *YOLO-VOC* modificado [24].

	Camadas	**Filtros**	**Tamanho**	**Entrada**	**Saída**
1	conv	32	3 x 3/1	240 x 80 x 3	240 x 80 x 32
2	max		2 x 2/1	240 x 80 x 32	120 x 40 x 32
3	conv	64	3 x 3/1	120 x 40 x 32	120 x 40 x 64
4	max		2 x 2/2	120 x 40 x 64	60 x 20 x 64
5	conv	128	3 x 3/1	60 x 20 x 64	60 x 20 x 128
6	conv	64	1 x 1/1	60 x 20 x 128	60 x 20 x 64
7	conv	128	3 x 3/1	60 x 20 x 64	60 x 20 x 128
8	max		2 x 2/2	60 x 20 x 128	30 x 10 x 128
9	conv	256	3 x 3/1	30 x 10 x 128	30 x 10 x 256
10	conv	128	1 x 1/1	30 x 10 x 256	30 x 10 x 128
11	conv	256	3 x 3/1	30 x 10 x 128	30 x 10 x 256
12	conv	512	3 x 3/1	30 x 10 x 256	30 x 10 x 512
13	conv	256	1 x 1/1	30 x 10 x 512	30 x 10 x 256
14	conv	512	3 x 3/1	30 x 10 x 256	30 x 10 x 512
15	conv	30	1 x 1/1	30 x 10 x 512	30 x 10 x 30
16	detecção				

4.2.5 Reconhecimento de Letras (*YOLO-VOC* Modificado)

Para o reconhecimento de letras, a mesma rede neural descrita na Subseção 4.2.4 foi utilizada, adaptando-se apenas o tamanho das camadas e a quantidade de filtros da última camada.

Tabela 4.5: Rede utilizada para reconhecimento de letras, utilizando *YOLO-VOC* modificado de [24].

	Camadas	**Filtros**	**Tamanho**	**Entrada**	**Saída**
1	conv	32	3 x 3/1	270 x 80 x 3	270 x 80 x 32
2	max		2 x 2/1	270 x 80 x 32	135 x 40 x 32
3	conv	64	3 x 3/1	135 x 40 x 32	120 x 40 x 64
4	max		2 x 2/2	135 x 40 x 64	67 x 20 x 64
5	conv	128	3 x 3/1	67 x 20 x 64	67 x 20 x 128
6	conv	64	1 x 1/1	67 x 20 x 128	67 x 20 x 64
7	conv	128	3 x 3/1	67 x 20 x 64	67 x 20 x 128
8	max		2 x 2/2	67 x 20 x 128	33 x 10 x 128
9	conv	256	3 x 3/1	33 x 10 x 128	33 x 10 x 256
10	conv	128	1 x 1/1	33 x 10 x 256	33 x 10 x 128
11	conv	256	3 x 3/1	33 x 10 x 128	33 x 10 x 256
12	conv	512	3 x 3/1	33 x 10 x 256	33 x 10 x 512
13	conv	256	1 x 1/1	33 x 10 x 512	33 x 10 x 256
14	conv	512	3 x 3/1	33 x 10 x 256	33 x 10 x 512
15	conv	155	1 x 1/1	33 x 10 x 512	33 x 10 x 155
16	detecção				

4.2.6 Reconhecimento de Números (*YOLO-VOC* Modificado)

Para essa rede, a arquitetura utilizada foi a da Subseção 4.2.4, tendo suas 4 primeiras camadas removidas, pois essa modificação não modifica o desempenho da rede, mas reduz o custo computacional do treinamento [19].

Tabela 4.6: Rede utilizada para reconhecimento de números, utilizando *YOLO-VOC* modificado de [19].

	Camadas	**Filtros**	**Tamanho**	**Entrada**	**Saída**
1	conv	128	3 x 3/1	42 x 26 x 3	42 x 26 x 128
2	conv	64	1 x 1/1	42 x 26 x 128	42 x 26 x 64
3	conv	128	3 x 3/1	42 x 26 x 64	42 x 26 x 128
4	max		2 x 2/1	42 x 26 x 128	21 x 13 x 128
5	conv	256	3 x 3/1	21 x 13 x 128	21 x 13 x 256
6	conv	128	1 x 1/1	21 x 13 x 256	21 x 13 x 128
7	conv	256	3 x 3/1	21 x 13 x128	21 x 13 x 256
8	conv	512	3 x 3/1	21 x 13 x 256	21 x 13 x 512
9	conv	256	1 x 1/1	21 x 13 x 512	21 x 13 x 256
10	conv	512	3 x 3/1	21 x 13 x 256	21 x 13 x 512
11	conv	75	1 x 1/1	21 x 13 x 512	21 x 13 x 75
12	detecção				

4.3 Pesos Pré-Treinados

Nos diversos testes realizados, três diferentes redes pré-treinadas foram utilizadas tanto para o *transfer learning* quando para o *fine-tuning*.

- UFPR: Treinamento de rede neural convolucional realizado por Laroca *et al.* [19] utilizando uma base de dados real anotada de placas brasileiras (Figura 4.6).
- Yolo V2: Treinamento realizado no desenvolvimento da arquitetura, a fim de se detectar ojetos de 17 classes (Disponível em [102]).
- *Darknet*:Desenvolvida para cumprir o desafio *ImageNet* [100] de 1000 classes (Disponível em [102]).

4.4 Cálculo da Acurácia

4.4.1 Localização da Placa Veicular na Imagem

No *framework* utilizado, uma saída é considerada válida se o nível de confiança dela é superior a 10% (o *threshold* foi configurado para 0.1). Para os casos em que a rede fornece como saída mais de uma previsão, apenas a previsão com o maior nível de confiança é considerada e a BB definida por ela tem sua interseção sobre união (*Intersection over Union - IoU*) calculada.

Uma placa é considerada encontrada corretamente quando a IoU da BB da placa identificada com a BB do GT da mesma placa é de pelo menos 70% ($IoU \geq 0.7$). A escolha desse valor se justifica pelo protocolo desenvolvido por Li *et al.* em 2017 [32] e seguido também pela revisão da literatura feita em 2018 por Xu *et al.* [103]. A Figura 4.9 ilustra diferentes resultados obtidos para cada IoU, enquanto a Figura 4.8 mostra graficamente como o cálculo é realizado.

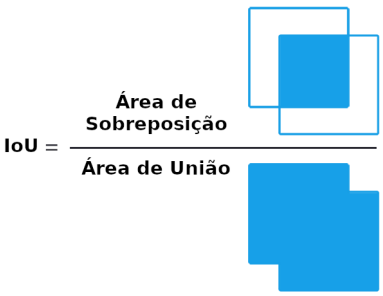


Figura 4.8: Cálculo da IoU. Modificado de [20].

(a)

(b)

(c)

Figura 4.9: Exemplos de *IoU* calculadas [1]: (a) *Intersection over Union* igual a 90,11%; (b) *Intersection over Union* igual a 75,44%; (c) *Intersection over Union* igual a 32,58%. A BB vermelha representa o GT da placa e a verde representa a BB definida pela rede neural.

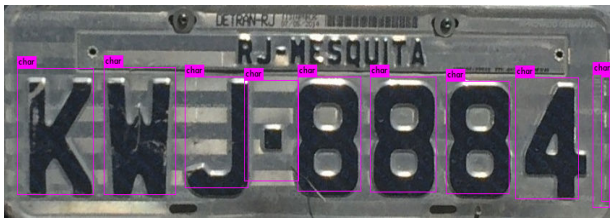
4.4.2 Segmentação e Reconhecimento de Caracteres

A métrica utilizada para as redes treinadas com o objetivo de segmentar os caracteres foi verificar se o resultado obtido continha exatamente 7 caracteres segmentados, qualquer saída das redes neurais que apresentassem uma quantidade diferente dessa foi classificada como uma previsão incorreta. A Figura 4.10 ilustra saídas obtidas dessas redes neurais.

Enquanto para o reconhecimento de caracteres a métrica é simples: a saída da rede neural deve corresponder ao caracter segmentado, caso contrário, a saída é considerada incorreta. Ressalta-se que as redes neurais responsáveis por reconhecer as letras e os números receberam como entrada (em todos os testes) os caracteres segmentados pela rede com maior acurácia da rede de segmentação dos mesmos.

Nas duas etapas dessa Subseção o nível de confiança mínimo calculado pela rede neural utilizado foi de 25% por apresentar os melhores resultados.

(a)



(b)

Figura 4.10: Exemplos de segmentações de caracteres realizadas [1]: (a) Exemplo de 7 caracteres segmentados corretamente; (b) Exemplo de quantidade incorreta de caracteres detectados.

Capítulo 5

Resultados e Discussões

Neste capítulo, os resultados para os treinamentos propostos nas Figuras 4.1 e 4.2 serão apresentados, reiterando-se que as acurácias foram calculadas em relação aos testes com uma base de dados real que não foi utilizada em qualquer etapa do treinamento, que foram realizados com bases artificiais divididas em conjunto de treinamento (75%) e teste (25%). Optou-se por padronizar o eixo horizontal de todos os gráficos como a representação de iterações e não de épocas (*epochs*) pois essa é nomenclatura utilizada pelo *framework*. A relação entre os termos é:

- **Época:** Quando uma rede neural recebeu todos os elementos do conjunto de treinamento como entrada e atualizou os pesos, diz-se que uma época foi finalizada.
- ***Batch*:** A quantidade de imagens utilizadas em um passo do treinamento é definida pelo tamanho do *batch*.
- **Iteração:** Número de passos realizados com o *batch* definido.

Exemplificando: o treinamento para localização de placa numa imagem realizado pelo grupo de pesquisa da UFPR [19] utilizou 1800 imagens de treinamento e teve 60 mil iterações. O *batch* utilizado foi igual a 64 (bem como no trabalho desenvolvido aqui), assim:

$$Epochs = \frac{(60000 \times 64)}{1800} \approx 2133. \tag{5.1}$$

Assim, sabe-se o tamanho de todas as bases de dados, *batches* e número de iterações de cada treinamento.

5.1 Redes Treinadas para encontrar uma Placa numa Imagem

5.1.1 Treinamento *Fast YOLO*: Pesos Iniciais Aleatórios *vs.* Pesos Iniciais UFPR

A Figura 5.3 ilustra a variação das acurácias das redes neurais em estágios intermediários de seus respectivos treinamentos, na qual verifica-se que os resultados das duas estratégias de treinamento assumiram valores próximos entre si após as 60 mil iterações propostas (uma diferença de 0,62% existe entre as acurácias finais).

Destaca-se ainda o comportamento exclusivamente decrescente do *transfer learning* realizado com pesos iniciais não aleatórios, pois durante o treinamento sua acurácia teve uma redução de 10,83%. É possível verificar também os baixos valores alcançados pelo treinamento indicado em verde na Figura 5.3 no qual o maior valor obtido se deu após 14 mil interações e foi igual a 15,63% que equivale a 176 placas localizadas corretamente, dada as 1126 imagens de placas da base de dados.

Ressalta-se que o treinamento realizado pela UFPR sem qualquer *transfer learning* com bases de dados sintéticas foi capaz de localizar corretamente 59,05% das placas da base de dados.

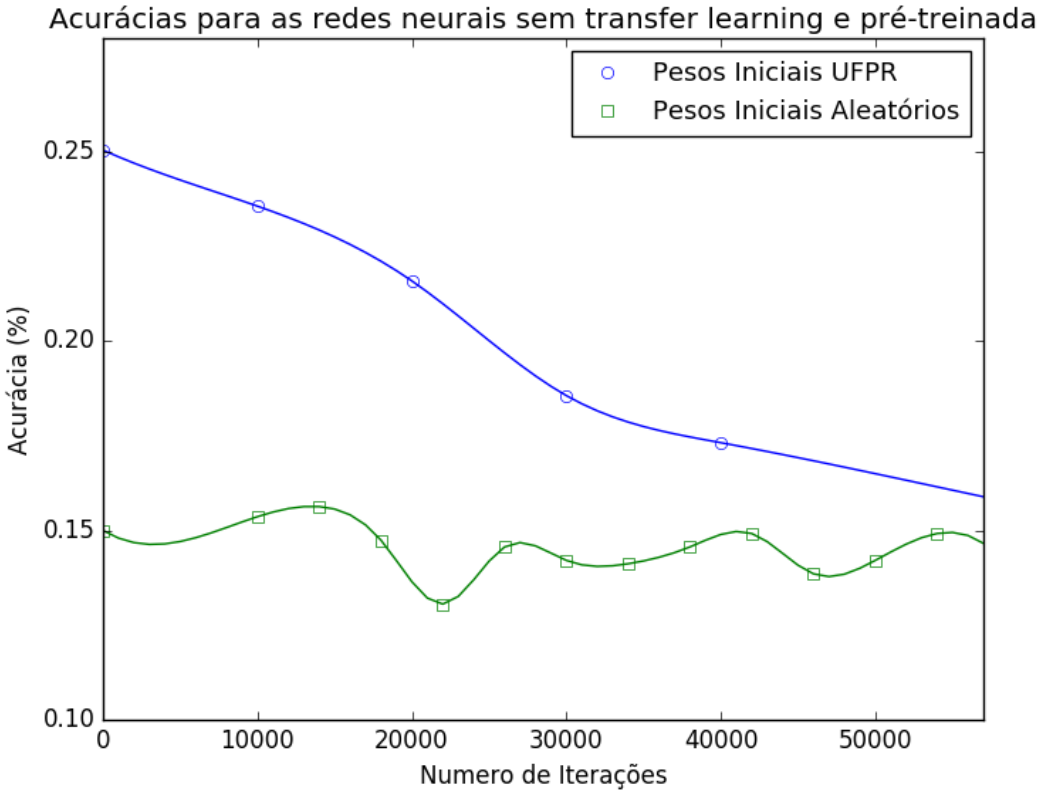


Figura 5.1: Comparação entre os treinamentos realizados sem *transfer learning* e com inicialização de pesos de outro treinamento.

5.1.2 Treinamento *Fast YOLO*: Camadas Congeladas e Variação de Decaimento dos Pesos (*Weight Decay - WD)*

Para os três treinamentos realizados nessa etapa (Figura 5.2) os valores de acurácia para os dois treinamentos com o mesmo número de camadas congeladas mostraram-se próximos, pois as médias dos estágios intermediários amostrados as acurácias para $WeightDecay = 0,0005$ e $WeightDecay = 0,00025$ foram iguais a 49,51% e 49,32%, respectivamente, evidenciando que para esse caso, a mudança no decaimento dos pesos não influenciou significativamente a acurácia dos modelos.

Verifica-se ainda uma diferença entre os valores obtidos para o mesmo valor de *Weight Decay* (com uma mudança na estratégia de congelamento de camadas), dado que a rede com 13 camadas congeladas acertou, em média, 557 das 1126 placas, enquanto a rede com 10 camadas congeladas localizou corretamente 474 placas veiculares (diferença de 7,37%).

O maior desvio padrão obtido foi para a arquitetura com 13 camadas congeladas sendo igual a 1,46% para o treinamento com o decaimento de pesos igual a 0,00025. Ao analisar-se a rede com o decaimento de 0.0005, o desvio padrão foi igual a 1,25%. Por fim, a única rede com 10 camadas congeladas apresentou um desvio padrão de 1,40%.

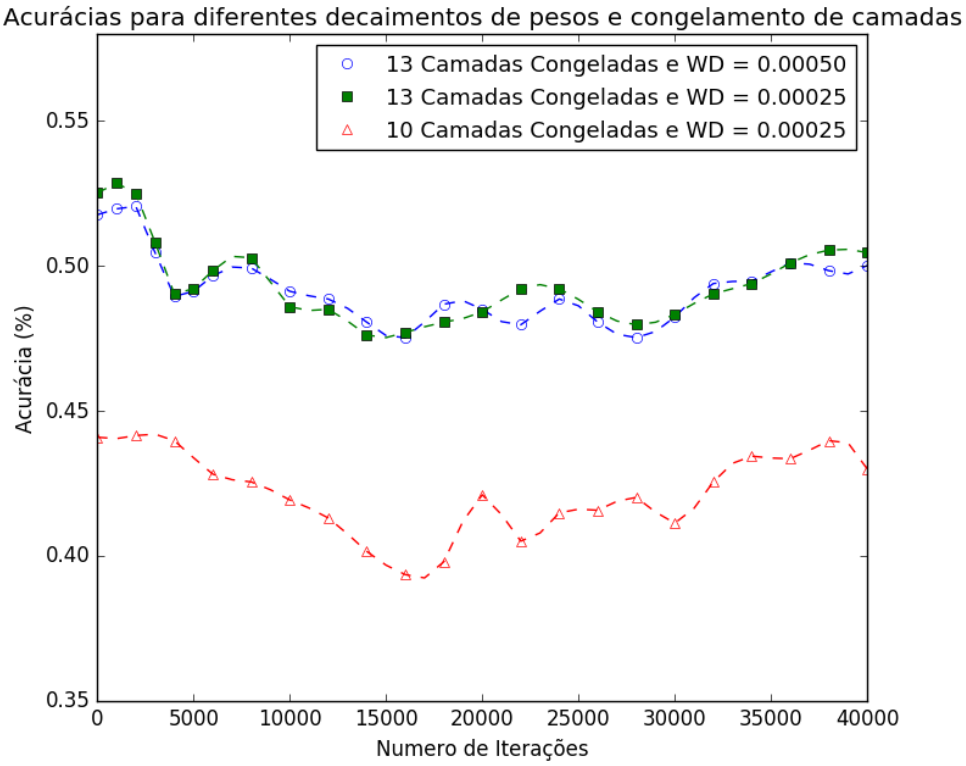


Figura 5.2: Influência do congelamento de camadas e da variação do decaimento dos pesos.

5.1.3 Treinamento *YOLO* v2 *vs. Darknet*

Comparando-se os resultados de ambos treinamentos realizados com pesos iniciais de redes neurais treinadas para reconhecer os mais variados objetos, nota-se que o *transfer learning* realizado com o *YOLO* v2 apresentou uma média de acurácia 2,23% maior que a arquitetura *Darknet* (26,26% e 24,03%, respectivamente) para as 30 mil iterações realizadas.

Mesmo com uma diferença relativamente inexpressiva entre as médias, as acurácias das redes analisadas começam a se distanciar a partir da 20.000ª iteração, chegando a apresentar uma diferença de aproximadamente 10% quando a acurácia do *YOLO* v2 atinge seu valor máximo e localiza 359 das 1126 placas corretamente (31,88%).

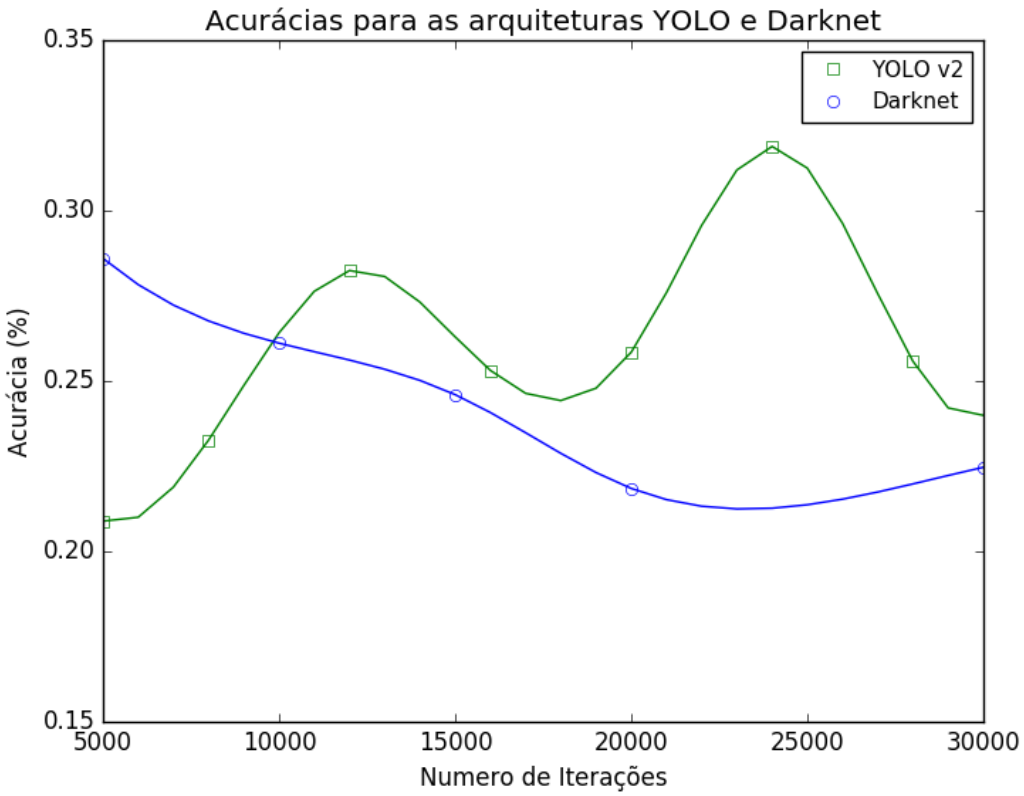


Figura 5.3: Comparação entre os treinamentos realizados com as arquiteturas *YOLO* v2 e *Darknet*. As primeiras cinco mil iterações não tiveram seus pesos intermediários registrados.

5.2 Redes Treinadas para segmentar os Caracteres de uma Placa

Considerando os treinamentos realizados para segmentar os caracteres, observa-se que apenas um deles superou o resultado obtido anteriormente com o treinamento da UFPR: o *transfer learning* realizado com os pesos iniciais da UFPR e mesma arquitetura fez com que a acurácia aumentasse em 2,58% fazendo com que o número de placas cujos caracteres foram segmentados corretamente aumentasse em 30 (passando de 854 para 884).

Tanto o treinamento *Fast YOLO* com 13 camadas congeladas (dado que a arquitetura da rede responsável por segmentar os caracteres é idêntica à responsável por localizar uma placa) quanto o treinamento com pesos iniciais aleatórios apresentaram desempenho inferior: respectivamente 760 e 283 placas reconhecidas, dentre 1126 (67,50 e 25,13%).

Tabela 5.1: Acurácias obtidas para os treinamentos das redes neurais de segmentação de caracteres.

Treinamento	**Acurácia (%)**
Fast YOLO (Pesos iniciais UFPR)	78,42
Treinamento UFPR	75,84
Fast YOLO (Pesos iniciais UFPR e 13 camadas congeladas)	67,50
Fast YOLO (Pesos iniciais aleatórios)	25,13

5.3 Redes Treinadas para reconhecer os Caracteres Segmentados

5.3.1 Reconhecimento de Letras

Essa foi a rede neural que apresentou o pior desempenho para o cenário com pesos iniciais aleatórios: apenas 307 letras dentre as 2652 foram reconhecidas corretamente (11,57%). Quanto às melhorias em relação ao treinamento inicial da UFPR, tanto o *transfer learning* quanto o *fine-tuning* fizeram a acurácia aumentar 1,09 e 0,27%, respectivamente.

Tabela 5.2: Acurácias obtidas para os treinamentos das redes neurais de reconhecimento de letras

Treinamento	**Acurácia (%)**
Fast YOLO (Pesos iniciais UFPR)	84,53
Fast YOLO (Pesos iniciais UFPR e 13 camadas congeladas)	83,71
Treinamento UFPR	83,44
Fast YOLO (Pesos iniciais aleatórios)	11,57

5.3.2 Reconhecimento de Números

Na etapa de reconhecimento dos números, o *fine-tuning* obteve uma acurácia maior que a rede que foi submetida ao *transfer learning*: a primeira reconheceu 10 números a mais que a segunda (3278 num conjunto de 3536 números - 92,76%).

O congelamento das 9 primeiras camadas da rede (que apresenta 12 camadas) fez com que a acurácia aumentasse em 2,49%. Assim, a rede foi capaz de reconhecer 88 caracteres adicionais em relação à quantidade previamente reconhecida pelo treinamento da UFPR (sem qualquer treinamento com bases de dados artificiais).

Considerando todas as redes treinadas com pesos iniciais aleatórios, a responsável por reconhecer os números apresentou o melhor resultado já que 2209 números de placas reais, dentre os 3536 avaliados (62,47%) foram classificados corretamente por uma rede neural que não recebeu números de placas capturadas em situações reais em qualquer etapa do seu treinamento.

Tabela 5.3: Acurácias obtidas para os treinamentos das redes neurais de reconhecimento de números

Treinamento	**Acurácia (%)**
Fast YOLO (Pesos iniciais UFPR e 9 camadas congeladas)	92,76%
Fast YOLO (Pesos iniciais UFPR)	92,44%
Treinamento UFPR	90,27%
Fast YOLO (Pesos iniciais aleatórios)	62,47%

Tabela 5.4: Acurácias obtidas para os treinamentos de segmentação e reconhecimento de caracteres.

Etapa ALPR/Treinamento	**Treinamento UFPR**	**Fast YOLO (Pesos iniciais UFPR)**	**Fast YOLO (Pesos iniciais aleatórios)**	**Fast YOLO (Pesos iniciais UFPR e camadas congeladas)**
Segmentação dos Caracteres	75,84%	78,42%	25,13%	67,50%
Reconhecimento de Letras	83,44%	84,53%	11,57%	83,71%
Reconhecimento de Números	90,27%	92,44%	62,47%	92,76%

5.4 Discussão dos Resultados

Inicialmente, analisando-se o desempenho de todas as redes neurais treinadas com pesos iniciais aleatórios e base de dados artificial, a Hipótese levantada em 1.3.3 foi parcialmente comprovada dado que 62 números em cada grupo de 100 (Tabela 5.3) foram reconhecidos por uma rede neural treinada exclusivamente com uma base de dados sintética, provando que não há a necessidade de se utilizar grandes bases de dados reais anotadas para essa tarefa de um sistema *ALPR*.

O resultado obtido para a rede responsável por reconhecer as letras treinada com a mesma arquitetura e pesos iniciais aleatórios apresentou uma acurácia 5 vezes menor. Justifica-se tal resultado pelo maior número de classes (que passa de 10 para 26), ambiguidade entre algumas letras (Q/O, M/N, D/O) constatada pelas previsões realizadas com nível de confiança de pelo menos 25% e pela quantidade reduzida de variações de cada letra, quando comparada com a quantidade de variações de cada número (20 e 50, respectivamente) como especificado na Subseção 4.1.5, definidos para que as redes de reconhecimento de letras e números tivessem o números de épocas similares.

A segmentação dos caracteres realizada pelo modelo inicializado com pesos aleatórios (ainda no contexto da Hipótese descrita em 1.3.3) foi capaz de segmentar corretamente os caracteres de

25,13% das placas reais. Considerando que o treinamento da rede de segmentar os caracteres foi realizado com uma base de dados sem variações (Fluxograma da Figura 4.2), o mesmo não extraiu *features* de placas capturadas em diferentes perspectivas ou ângulos, restringindo assim sua capacidade de extrair corretamente os 7 caracteres esperados, ainda assim, nota-se uma razoável capacidade de aprendizado da rede com esse resultado.

Além disso considerando a acurácia obtida com o treinamento utilizando-se pesos iniciais aleatórios da rede responsável por localizar uma placa veicular numa imagem (máximo de 15,63%, como explicitado na Subseção 5.1.1) classifica-se como um resultado promissor, que pode ser melhorado a partir de um treinamento com uma base de dados maior que inclua maiores variações e objetos numa cena, ou mesmo numa rede neural que seja treinada por mais *epochs*.

Ao analisar-se a influência da utilização de pesos obtidos por treinamentos anteriores com placas reais, a Hipótese elaborada em 1.3.4 é totalmente confirmada pelos resultados obtidos para todas as redes neurais, porém com aspectos diferentes, a influência nas redes de segmentação dos caracteres (Tabela 5.1), reconhecimento de letras (Tabela 5.2) e na rede responsável por classificar os números (Tabela 5.3) é positiva, na medida em que para todas essas etapas houve um aumento da acurácia causado pelo *transfer learning*.

A única rede inicializada com pesos iniciais de um treinamento realizado com placas reais que não apresentou melhorias significativas foi a responsável por localizar uma placa na imagem, de sorte que houve uma redução de 6,21% na acurácia quando compara-se a obtida pelo treinamento UFPR sem qualquer *transfer learning* (igual a 59,05% como citado na Subseção 5.1.1) ao valor máximo obtido pelo *fine-tuning* no cenário com 13 camadas congeladas e decaimento de pesos igual a 0.00025 (52,84% como ilustrado na Subseção 5.1.2).

Salientando-se ainda a análise do efeito de aplicar a estratégia de congelar camadas, é possível verificar a maior variação de acurácia obtida nesse trabalho: o *transfer learning* com pesos iniciais da UFPR e todas as camadas livres alcançou uma acurácia de 14,20% ao fim das iterações(Subseção 5.1.1), enquanto a mesma arquitetura com 13 camadas congeladas e decaimento de pesos igual a 0,0005 alcançou uma acurácia média de 49,51% (Subseção 5.1.2) na etapa de localizar uma placa. Evidenciando que o *fine-tuning* realizado ocasionou num aumento de 35,31% na acurácia do modelo.

Por fim, os resultados evidenciados na Subseção 5.1.3 mostram que redes neurais profundas previamente treinadas para reconhecer os mais diferentes objetos podem ser utilizadas como ponto de partida para um treinamento específico, sabendo que a estratégia já foi utilizada para reconhecer plantas [104], imagens médicas [105] e placas de carros norueguesas [22].

A estratégia fez com que a acurácia máxima da arquitetura *YOLO* v2 fosse igual a 31,88% (Subseção 5.1.3) com apenas 24000 iterações, mostrando que as *features* extraídas de outros treinamentos (informações de bordas, cores, posição na imagem, etc.) podem ser úteis ao se desenvolver um modelo para reconhecer um número menor de classes.

Capítulo 6

Conclusões

Assim, o trabalho descrito nesse manuscrito propôs uma abordagem para a avaliação tanto do treinamento com bases artificiais quanto do *fine-tuning* e *transfer learning* de redes neurais desenvolvidas para sistemas *ALPR*. A utilização de bases de dados sintética, criadas com o objetivo de se evitar a anotação manual de base de dados reais, mostrou-se como uma estratégia a ser analisada e possivelmente implementada num sistema de reconhecimento de placas veiculares ao aumentar a acurácia de duas das três etapas (Subseções 5.2 e 5.3) enumeradas em 1.1, atingindo o Objetivo Geral proposto em 1.3.1.

O Objetivo Específico previamente estabelecido (1.3.2) possibilitou verificar a capacidade de redes neurais e arquiteturas desenvolvidas para reconhecer variados objetos em reconhecer uma classe específica, obtendo uma acurácia de 31,88% com as iterações realizadas. Confirmando assim, que as redes neurais profundas podem ser utilizadas para tarefas específicas a partir de um retreinamento realizado com os parâmetros corretos, incluindo a utilização de bases sintéticas.

Embora os resultados obtidos com os treinamentos que utilizaram exclusivamente bases de dados artificiais não tenham obtido valores próximos àqueles com bases reais, os mesmos indicam que existe uma considerável capacidade de generalização da arquitetura utilizada com o aprendizado adquirido.

Por fim, o congelamento de camadas de uma rede neural também se mostrou como uma relevante estratégia, dado que o *fine-tuning* ocasionou em significantes melhorias na saída dos sistemas (5.1.2), confirmando a Hipótese descrita em 1.3.4 e indicando ainda que a influência das placas sintéticas no treinamento é positiva.

6.1 Trabalhos Futuros

Analisando-se os resultados obtidos, verifica-se que a utilização de bases de dados sintéticas para o reconhecimento de placas veiculares no padrão brasileiro pode melhorar o desempenho de algumas etapas do processo.

A continuidade desse trabalho poderia ser dada ao se utilizar base de dados sintéticas ainda

maiores, ou mesmo adicionando variações de perspectiva, rotação e ruído para o treinamento da rede responsável por segmentar os caracteres. Uma outra análise pode ser feita ainda utilizando *frameworks* diferentes para as etapas de localizar uma placa (reconhecimento de objetos) e segmentar e reconhecer os caracteres (ferramentas de *OCR*), não realizado aqui por não ser coberto pelo escopo do trabalho e pela disponibilidade de pesos pré-treinados para o padrão de placas brasileiras.

Ressalta-se ainda que apenas a arquitetura *Fast YOLO* foi avaliada tanto para o *transfer learning* quanto para o *fine-tugmail.comning*, motivada pela quantidade de treinamentos realizados e recursos disponíveis. Assim, uma arquitetura mais complexa como a recém lançada *YOLO* v3 que apresenta 106 camadas [102] poderia ser testada.

O trabalho desenvolvido nesse manuscrito considerando o formato vigente de placas brasileiras poderá ser naturalmente utilizado, recorrendo-se aos seus resultados como parâmetros iniciais para o modelo que o Brasil passará a adotar nos próximos meses. A inviabilidade de tal alteração nesse projeto justifica-se pela inexistência de bases de dados reais anotadas com o padrão de placas veiculares do novo padrão (Mercosul - Mercado Comum do Sul).

REFERÊNCIAS BIBLIOGRÁFICAS

[1] PLATES BR - Git Hub. https://github.com/openalpr/plates_br/tree/master/images. Acessado em: 10-11-2018.

[2] GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. http://www.deeplearningbook.org.

[3] A Practical Introduction to Deep Learning with Caffe and Python. http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/, note = Acessado em: 12-11-2018.

[4] DUMOULIN, V.; VISIN, F. A guide to convolution arithmetic for deep learning. *arXiv:1603.07285v2*, 2018.

[5] HIEN, D. H. T. *A Guide to Receptive Field Arithmetic for Convolutional Neural Networks*. https://medium.com/syncedreview/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks-42f33d4378e0. Acessado em: 14-11-2018.

[6] MAX-POOLING / Pooling. https://computersciencewiki.org/index.php/Max-pooling_/_Pooling. Acessado em: 14-11-2018.

[7] STANFORD cs231n Universidade de. *CS231n Convolutional Neural Networks for Visual Recognition*. hhttp://cs231n.github.io/convolutional-networks/#pool. Acessado em: 14-11-2018.

[8] (UNSW), S. Z. *Introduction to CNN*. https://www.slideshare.net/ShuaiZhang33/lg-cnn20170213. Acessado em: 14-11-2018.

[9] CUBUK, E. D. et al. Autoaugment:learning augmentation policies from data. *arXiv:1805.09501v2*, Out 2018.

[10] ZHU, J.-Y. et al. Unpaired image-to-image translation using cycle-consistent adversarial networks. *International Conference on Computer Vision*, Aug 2017.

[11] GENERALIZATION and Overfitting. https://wp.wwu.edu/machinelearning/2017/01/22/generalization-and-overfitting/. Acessado em: 14-11-2018.

[12] GIRSHICK, R. B. et al. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013. Disponível em: <http://arxiv.org/abs/1311.2524>.

[13] HE, K. et al. Spatial pyramid pooling in deep convolutional networks for visual recognition. *CoRR*, abs/1406.4729, 2014. Disponível em: <http://arxiv.org/abs/1406.4729>.

[14] LIN, T. et al. Feature pyramid networks for object detection. *CoRR*, abs/1612.03144, 2016. Disponível em: <http://arxiv.org/abs/1612.03144>.

[15] HE, K. et al. Mask R-CNN. *CoRR*, abs/1703.06870, 2017. Disponível em: <http://arxiv.org/abs/1703.06870>.

[16] IMAGE segmentation with Mask R-CNN. `https://medium.com/@jonathan_hui/image-segmentation-with-mask-r-cnn-ebe6d793272`. Acessado em: 23-11-2018.

[17] REDMON, J. et al. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. Disponível em: <http://arxiv.org/abs/1506.02640>.

[18] LIU, W. et al. Ssd: Single shot multibox detector. *arXiv:1512.02325v5*, 2016.

[19] LAROCA, R. et al. A robust real-time automatic license plate recognition based on the yolo detector. In: *2018 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2018. p. 1–10. ISSN 2161-4407.

[20] YOLO: Real-Time Object Detection. `https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/`. Acessado em: 06-08-2018.

[21] PATEL, C.; SHAH, D.; PATEL, A. Article: Automatic number plate recognition system (anpr): A survey. *International Journal of Computer Applications*, v. 69, n. 9, p. 21–33, May 2013.

[22] JORGENSEN, H. Automatic license plate recognition using deep learning techniques. *https://brage.bibsys.no/xmlui/handle/11250/2467209*, 2017.

[23] REDMON, J. *ImageNet Classification*. `https://pjreddie.com/darknet/imagenet/`. Acessado em: 23-11-2018.

[24] SILVA, S. M.; JUNG, C. R. Real-time brazilian license plate detection and recognition using deep convolutional neural networks. In: *2017 30th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*. [S.l.: s.n.], 2017. p. 55–62. ISSN 2377-5416.

[25] KIM, J. et al. License plate detection and recognition algorithm for vehicle black box. In: *2017 International Automatic Control Conference (CACS)*. [S.l.: s.n.], 2017. p. 1–6.

[26] DAVIES, P.; EMMOTT, N.; AYLAND, N. License plate recognition technology for toll violation enforcement. In: *IEE Colloquium on Image Analysis for Transport Applications*. [S.l.: s.n.], 1990. p. 7/1–7/5.

[27] SEIBEL, H.; GOLDENSTEIN, S.; ROCHA, A. Eyes on the target: Super-resolution and license-plate recognition in low-quality surveillance videos. *IEEE Access*, v. 5, p. 20020–20035, 2017. ISSN 2169-3536.

[28] OWAYJAN, M. et al. Lebanese license plate recognition for driving tickets automation system. *20th LAAS International Science Conference (LAAS 14)*, Mar 2014.

[29] BHARGAVA, K.; GOYAL, D. A video surveillance system for speed detection of vehicles and law enforcement using automatic number plate recognition. *International journal of research in Computer Engineering and Electronics*, v. 3, Jan 2014. ISSN 2319-376X.

[30] DU, S. et al. Automatic license plate recognition (alpr):a state-of-the-art review. *IEEE Transactions on Circuits and systems for video technology*, v. 84, p. 311–325, fev. 2003.

[31] ANAGNOSTOPOULOS, C.-N. E. et al. License plate recognition from still images and video sequences: A survey. *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS*, v. 9, 2008.

[32] LI, H.; WANGY, P.; SHEN, C. Towards end-to-end car license plates detection and recognition with deep neural networks. *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS*, Set 2017.

[33] NATHAN, V. S. L.; S, R. J. andKamakshi P. New approaches for license plate recognition system. *Proc. Int. Conf. Intell. Sens. Inform. Process.*, p. 149–152, 2004.

[34] MATAS, J.; ZIMMERMANN, K. Unconstrained licence plate and text localization and recognition. *Proceedings of the 17th International Conference on Pattern Recognition*, p. 225–230, 2005.

[35] HONGLIANG, B.; CHANGPING, L. A hybrid license plate extraction method based on edge statistics and morphology. *Proceedings of the 17th International Conference on Pattern Recognition*, v. 2, p. 831–834, 2004.

[36] BUSCH, C.; DORNER, C. F. R.; ZIEGLER, H. Feature based recognition of traffic video streams for online route tracing. *Vehicular Technology Conference*, v. 3, p. 1790–1794, 1998.

[37] XU, X. et al. A method of multi-view vehicle license plates location based on rectangle features. *International Conference on Signal Processing*, v. 3, p. 16–20, 2006.

[38] PAN, M.-S.; JUN-BIAO; XIAO, Y.-H. Vehicle license plate character segmentation. *International Journal of Automation and Computing*, v. 5, p. 102–432, out. 2008.

[39] SARFRAZ, M.; AHMED, M. J.; GHAZI, S. A. Saudi abarian license plate recognition system. *International Conference on Geometric Modeling and Graphics*, p. 36–41, 2003.

[40] RAHMAN, C. A.; BADAWY, W.; RADMANESH, A. A real time vehicle's license plate recognition system. *International Conference on Geometric Modeling and Graphics*, p. 163–166, 2003.

[41] GRYS, B. T. et al. Machine learning and computer vision approaches for phenotypic profiling. *The Journal of cell biology*, v. 216(1), p. 65–71, 2017.

[42] YAMASHITA, R.; NISHIO, M.; TOGASHI, R. K. G. D. K. Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, v. 9(4), p. 611–629, Aug 2018, doi:10.1007/s13244-018-0639-9.

[43] KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: PEREIRA, F. et al. (Ed.). *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012. p. 1097–1105. Disponível em: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.

[44] YAMASHITA, R.; NISHIO, M.; TOGASHI, R. K. G. D. K. Fish species identification using a convolutional neural network trained on synthetic data. *ICES Journal of Marine Science*, 2018, doi:10.1093/icesjms/fsy147.

[45] DENATRAN. Resolução 231. *https://www.denatran.gov.br/download/Resolucoes/RESOLUCAO_231.pdf*, 15 de Março 2007.

[46] REDMON, J.; FARHADI, A. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.

[47] KASPAROV perde uma partida de xadrez para um computador. `https://seuhistory.com/hoje-na-historia/kasparov-perde-uma-partida-de-xadrez-para-um-computador`, note = Acessado em: 06-08-2018.

[48] LECUN, Y.; BENGIO, Y.; HINTON, G. *Deep learning*. Jun 2015.

[49] HAASWIJK, W. et al. Deep learning for logic optimization algorithms. In: *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. [S.l.: s.n.], 2018. p. 1–4. ISSN 2379-447X.

[50] KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. *NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems*, v. 1, p. 1097–1105, Dez 2012.

[51] LECUN, Y.; BENGIO, Y. The handbook of brain theory and neural networks. In: ARBIB, M. A. (Ed.). Cambridge, MA, USA: MIT Press, 1998. cap. Convolutional Networks for Images, Speech, and Time Series, p. 255–258. ISBN 0-262-51102-9. Disponível em: <http://dl.acm.org/citation.cfm?id=303568.303704>.

[52] HUBEL, D.; WIESEL, T. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *Journal of physiology*, v. 160,1, p. 106–154, 1962.

[53] BOJARSKI, M. et al. Visualbackprop: visualizing cnns for autonomous driving. *CoRR*, abs/1611.05418, 2016. Disponível em: <http://arxiv.org/abs/1611.05418>.

[54] LECUN, Y. et al. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, v. 1, n. 4, p. 541–551, Dec 1989. ISSN 0899-7667.

[55] SCHERER, D.; MüLLER, A.; BEHNKE, S. Evaluation of pooling operations in convolutional architectures for object recognition. In: . [S.l.: s.n.], 2010. p. 92–101.

[56] SCARDAPANE, S. et al. Complex-valued neural networks with non-parametric activation functions. *CoRR*, abs/1802.08026, 2018. Disponível em: <http://arxiv.org/abs/1802.08026>.

[57] PAN, S. J.; YANG, Q. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, v. 22, n. 10, p. 1345–1359, Oct 2010. ISSN 1041-4347.

[58] BLITZER, J.; DREDZE, M.; PEREIRA, F. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In: *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*. Association for Computational Linguistics, 2007. p. 440–447. Disponível em: <http://aclweb.org/anthology/P07-1056>.

[59] KWASIGROCH, A.; MIKOŁAJCZYK, A.; GROCHOWSKI, M. Deep convolutional neural networks as a decision support tool in medical problems – malignant melanoma case study. In: *Trends in Advanced Intelligent Control, Optimization and Automation*. [S.l.]: Springer International Publishing, 2017. p. 848–856. ISBN 978-3-319-60699-6.

[60] KWASIGROCH, A.; MIKOŁAJCZYK, A.; GROCHOWSKI, M. Deep neural networks approach to skin lesions classification — a comparative analysis. In: *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*. [S.l.: s.n.], 2017. p. 1069–1074.

[61] GOODFELLOW, I. J. et al. Generative adversarial networks. *https://arxiv.org/abs/1606.03498*, Aug 2017.

[62] BULLINARIA, J. A. *Bias and Variance, Under-Fitting and Over-Fitting*. `http://www.cs.bham.ac.uk/~jxb/INC/l9.pdf`. Acessado em: 10-11-2018.

[63] HONGLIANG, B.; CHANGPING, L. A hybrid license plate extraction method based on edge statistics and morphology. In: *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* [S.l.: s.n.], 2004. v. 2, p. 831–834 Vol.2. ISSN 1051-4651.

[64] LUO, L. et al. An efficient method of license plate location. In: *2009 First International Conference on Information Science and Engineering*. [S.l.: s.n.], 2009. p. 770–773. ISSN 2160-1283.

[65] KANAYAMA, K. et al. Development of vehicle-license number recognition system using real-time image processing and its application to travel-time measurement. In: *[1991 Proceedings] 41st IEEE Vehicular Technology Conference*. [S.l.: s.n.], 1991. p. 798–804. ISSN 1090-3038.

[66] KAMAT, V.; GANESAN, S. An efficient implementation of the hough transform for detecting vehicle license plates using dsp's. In: *Proceedings Real-Time Technology and Applications Symposium*. [S.l.: s.n.], 1995. p. 58–59. ISSN 1080-1812.

[67] BUSCH, C. et al. Feature based recognition of traffic video streams for online route tracing. In: *VTC '98. 48th IEEE Vehicular Technology Conference. Pathway to Global Wireless Revolution (Cat. No.98CH36151)*. [S.l.: s.n.], 1998. v. 3, p. 1790–1794 vol.3. ISSN 1090-3038.

[68] SANYUAN, Z.; MINGLI, Z.; XIUZI, Y. Car plate character extraction under complicated environment. In: *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*. [S.l.: s.n.], 2004. v. 5, p. 4722–4726 vol.5. ISSN 1062-922X.

[69] AL-GHAILI, A. M. et al. A new vertical edge detection algorithm and its application. In: *2008 International Conference on Computer Engineering Systems*. [S.l.: s.n.], 2008. p. 204–209.

[70] DEB, K.; CHAE, H.-U.; JO, K.-H. Vehicle license plate detection method based on sliding concentric windows and histogram. *JCP*, v. 4, p. 771–777, 2009.

[71] ANAGNOSTOPOULOS, C. N. E. et al. A license plate-recognition algorithm for intelligent transportation system applications. *IEEE Transactions on Intelligent Transportation Systems*, v. 7, n. 3, p. 377–392, Sept 2006. ISSN 1524-9050.

[72] SHI, X.; ZHAO, W.; SHEN, Y. Automatic license plate recognition system based on color image processing. In: *Proceedings of the 2005 International Conference on Computational Science and Its Applications - Volume Part IV*. Berlin, Heidelberg: Springer-Verlag, 2005. (ICCSA'05), p. 1159–1168. ISBN 3-540-25863-9, 978-3-540-25863-6. Disponível em: <http://dx.doi.org/10.1007/11424925$_1$21>.

[73] CHANG, S.-L. et al. Automatic license plate recognition. *Trans. Intell. Transport. Sys.*, IEEE Press, Piscataway, NJ, USA, v. 5, n. 1, p. 42–53, mar. 2004. ISSN 1524-9050. Disponível em: <https://doi.org/10.1109/TITS.2004.825086>.

[74] YANG, Y.-Q. et al. A vehicle license plate recognition system based on fixed color collocation. In: *2005 International Conference on Machine Learning and Cybernetics*. [S.l.: s.n.], 2005. v. 9, p. 5394–5397 Vol. 9. ISSN 2160-133X.

[75] MATAS, J.; ZIMMERMANN, K. Unconstrained licence plate and text localization and recognition. In: *Proceedings. 2005 IEEE Intelligent Transportation Systems, 2005.* [S.l.: s.n.], 2005. p. 225–230. ISSN 2153-0009.

[76] CHO, B. K. et al. License plate extraction method for identification of vehicle violations at a railway level crossing. *International Journal of Automotive Technology*, v. 12, n. 2, p. 281–289, Apr 2011. ISSN 1976-3832. Disponível em: <https://doi.org/10.1007/s12239-011-0033-9>.

[77] LEE, E. R.; KIM, P. K.; KIM, H. J. Automatic recognition of a car license plate using color image processing. In: *Proceedings of 1st International Conference on Image Processing*. [S.l.: s.n.], 1994. v. 2, p. 301–305 vol.2.

[78] MIYAMOTO, K. et al. Vehicle license-plate recognition by image analysis. In: *Proceedings IECON '91: 1991 International Conference on Industrial Electronics, Control and Instrumentation*. [S.l.: s.n.], 1991. p. 1734–1738 vol.3.

[79] NUKANO, T.; FUKUMI, M.; KHALID, M. Vehicle license plate character recognition by neural networks. In: *Conference: Intelligent Signal Processing and Communication Systems*. [S.l.: s.n.], 2004. p. 771 – 775.

[80] DUAN, T. D. et al. Building an automatic vehicle license-plate recognition system. In: *n Proc. Int. Conf. Comput. Sci. RIVF*. [S.l.: s.n.], 2005. p. 59 – 63.

[81] PALIY, I. et al. Approach to recognition of license plate numbers using neural networks. In: *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*. [S.l.: s.n.], 2004. v. 4, p. 2965–2970 vol.4. ISSN 1098-7576.

[82] SMITH, R. An overview of the tesseract ocr engine. In: *Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 02*. Washington, DC, USA: IEEE Computer Society, 2007. (ICDAR '07), p. 629–633. ISBN 0-7695-2822-8. Disponível em: <http://dl.acm.org/citation.cfm?id=1304596.1304846>.

[83] RASHEED, S.; NAEEM, A.; ISHAQ, O. Automated number plate recognition using hough lines and template matching. *Proceedings of the World Congress on Engineering and Computer Science*, v. 1, Out 2008.

[84] KIM, K. K. et al. Learning-based approach for license plate recognition. In: *Neural Networks for Signal Processing X. Proceedings of the 2000 IEEE Signal Processing Society Workshop (Cat. No.00TH8501)*. [S.l.: s.n.], 2000. v. 2, p. 614–623 vol.2. ISSN 1089-3555.

[85] SHUANG-TONG, T.; WEN-JU, L. Number and letter character recognition of vehicle license plate based on edge hausdorff distance. In: *Sixth International Conference on Parallel and Distributed Computing Applications and Technologies (PDCAT'05)*. [S.l.: s.n.], 2005. p. 850–852. ISSN 2379-5352.

[86] LI, H.; SHEN, C. Reading car license plates using deep convolutional neural networks and lstms. *CoRR*, abs/1601.05610, 2016. Disponível em: <http://arxiv.org/abs/1601.05610>.

[87] JIAO, J.; YE, Q.; HUANG, Q. A configurable method for multi-style license plate recognition. *Pattern Recogn.*, Elsevier Science Inc., New York, NY, USA, v. 42, n. 3, p. 358–369, mar. 2009. ISSN 0031-3203. Disponível em: <http://dx.doi.org/10.1016/j.patcog.2008.08.016>.

[88] THE PASCAL Visual Object Classes Homepage. `http://host.robots.ox.ac.uk/pascal/VOC/`. Acessado em: 21-11-2018.

[89] ZHAO, Z.-Q. et al. Object detection with deep learning: A review. *CoRR*, abs/1807.05511, 2018.

[90] UIJLINGS, J. R. R. et al. Selective search for object recognition. *International Journal of Computer Vision*, v. 104, n. 2, p. 154–171, Sep 2013. ISSN 1573-1405. Disponível em: <https://doi.org/10.1007/s11263-013-0620-5>.

[91] GRAUMAN, K.; DARRELL, T. The pyramid match kernel: discriminative classification with sets of image features. In: *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*. [S.l.: s.n.], 2005. v. 2, p. 1458–1465 Vol. 2. ISSN 1550-5499.

[92] GIRSHICK, R. B. Fast R-CNN. *CoRR*, abs/1504.08083, 2015. Disponível em: <http://arxiv.org/abs/1504.08083>.

[93] REN, S. et al. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015. Disponível em: <http://arxiv.org/abs/1506.01497>.

[94] FELZENSZWALB, P. F. et al. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 32, n. 9, p. 1627–1645, Sept 2010. ISSN 0162-8828.

[95] ITSEEZ. *Open Source Computer Vision Library*. 2015. `https://github.com/itseez/opencv`.

[96] XIAO, J. et al. Sun database: Large-scale scene recognition from abbey to zoo. *IEEE Conference on Computer Vision and Pattern Recognition*, abs/1703.06870, 2010.

[97] SHREERAMAN. *Data Augmentation for Object Detection(YOLO)*. `https://github.com/srp-31/Data-Augmentation-for-Object-Detection-YOLO-`.

[98] REDMON, J. *Darknet: Open Source Neural Networks in C*. `https://pjreddie.com/darknet/`. Acessado em: 23-11-2018.

[99] HAMORI, S. et al. *Ensemble Learning or Deep Learning? Application to Default Risk Analysis*. [S.l.], jan. 2018. Disponível em: <https://ideas.repec.org/p/koe/wpaper/1802.html>.

[100] LARGE Scale Visual Recognition Challenge. `http://image-net.org/challenges/LSVRC/`. Acessado em: 23-11-2018.

[101] GONçALVES, G. R.; MENOTTI, D.; SCHWARTZ, W. R. License plate recognition based on temporal redundancy. In: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. [S.l.: s.n.], 2016. p. 2577–2582. ISSN 2153-0017.

[102] YOLO: Real-Time Object Detection. `https://pjreddie.com/darknet/yolo/`. Acessado em: 06-08-2018.

[103] XU, Z. et al. Towards end-to-end license plate detection and recognition: A large dataset and baseline. In: *The European Conference on Computer Vision (ECCV)*. [S.l.: s.n.], 2018.

[104] REYES, A. K.; CAICEDO, J. C.; CAMARGO, J. E. Fine-tuning deep convolutional networks for plant recognition. In: *CLEF*. [S.l.: s.n.], 2015.

[105] TAJBAKHSH, N. et al. Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE Transactions on Medical Imaging*, v. 35, n. 5, p. 1299–1312, May 2016. ISSN 0278-0062.

ANEXOS

I. CONTEÚDO DO CD

II. HARDWARE

Os computadores utilizados no treinamento das redes neurais foram:

Computador do Laboratório de Imagens, Sinais e Áudio - LISA/CIC-UnB

CPU: Intel ®Core^{TM} i7-930 CPU @ 2.80GHz
Memória CPU: 5957 MiB
GPU: GeForce GTX 580
CUDA Cores: 512
Memória total GPU: 1536 MB

Computador do Laboratório de Sistemas Integrados e Concorrentes - LAICO/CIC-UnB

CPU: Intel ®Core^{TM} i3-7100 CPU @ 3.90GHz
Memória CPU: 7843 MiB
GPU: GeForce GTX 1050 Ti
CUDA Cores:768
Memória total GPU: 4096 MB